

*Matematyka stosowana*

# Obliczenia naukowe

Piotr Krzyżanowski  
przykry@mimuw.edu.pl  
<http://www.mimuw.edu.pl/~przykry>

Uniwersytet Warszawski, 2012



**Streszczenie.** W praktyce nie sposób uprawiać matematyki stosowanej bez wykorzystania metod numerycznych i narzędzi programistycznych do rozwiązywania pojawiających się tam zagadnień.

Na kolejnych zajęciach będziemy rozważać konkretne zadania obliczeniowe, jakie pojawiają się w realnych zastosowaniach: w elektrotechnice, biologii, telekomunikacji, chemii, rolnictwie, astrofizyce, itd. Jest tradycją tego przedmiotu, że co roku pojawiają się w nim nieco inne zadania, dlatego omówione tutaj przykłady należy traktować jako pewien zestaw, hmmm... przykładowy.

Wersja internetowa wykładu:

<http://mst.mimuw.edu.pl/lecture.php?lecture=ona>

(może zawierać dodatkowe materiały)



Niniejsze materiały są dostępne na [licencji Creative Commons 3.0 Polska](#):  
*Uznanie autorstwa — Użycie niekomercyjne — Bez utworów zależnych.*

---

Copyright © P.Krzyżanowski, Uniwersytet Warszawski, Wydział Matematyki, Informatyki i Mechaniki, 2012.  
Niniejszy plik PDF został utworzony 21 marca 2012.

---



Projekt współfinansowany przez Unię Europejską w ramach  
[Europejskiego Funduszu Społecznego](#).



---

Skład w systemie L<sup>A</sup>T<sub>E</sub>X, z wykorzystaniem m.in. pakietów beamer oraz listings. Szablony podręcznika i prezentacji:  
Piotr Krzyżanowski; koncept: Robert Dąbrowski.

# Spis treści

<b>Wprowadzenie</b>	5
<b>1. Potrzeba i ból obliczeń numerycznych</b>	7
1.1. Przegląd metod i narzędzi	8
1.1.1. Rachunek symboliczny kontra obliczenia numeryczne	8
1.1.2. Środowiska komputerowe wspomagające pracę matematyczną	10
<b>2. Octave: jak w nim pracować</b>	19
<b>3. Funkcja odwrotna</b>	20
3.1. Doprecyzowanie zadania	20
3.2. Transformacja zadania do wygodniejszej postaci	22
3.3. A może wystarczy po prostu... wziąć większy młotek?	23
<b>4. Reakcja enzymatyczna</b>	25
4.1. Równanie Michaelisa–Menten	25
4.1.1. Nietrafione uproszczenia	25
4.1.2. Rozwiązanie nieliniowego zadania najmniejszych kwadratów	29
4.2. Różniczkowy model łańcucha reakcji enzymatycznych	31
<b>5. Hodowla zwierząt</b>	34
5.1. Dyskusja problemu	35
5.2. Implementacja	37
5.3. Wykorzystanie specyfiki zadania	37
5.4. Przypadek dużego $n$	38
<b>6. Charakterystyka pracy transformatora</b>	42
6.1. Bezstresowe rozwiązanie problemu	42
6.1.1. Próba weryfikacji rozwiązania	43
6.2. Czy interpretacja wyniku jest poprawna?	44
6.2.1. Jak dobrać dobrą rozdzielczość?	46
6.3. Eksperymenty z parametrami układu	48
6.4. Uwagi i uzupełnienia	48
<b>7. Linie i okręgi</b>	49
7.1. Dopasowanie prostej	49
7.1.1. Uproszczenia i analiza problemu	50
7.1.2. Implementacja	51
7.2. Dopasowanie okręgu	52
7.2.1. Atak na wprost	52
7.2.2. Zmiana sformułowania zadania	53
7.2.3. Wybór przybliżenia początkowego jako nowe zadanie	54
7.3. Uwagi i uzupełnienia	56
<b>8. Stacjonarne równanie dyfuzji</b>	58
8.1. Laplasjan: dyskretyzacja metodą różnic skończonych	58
8.1.1. Dyskretyzacja jednowymiarowego laplasjanu	59
8.1.2. Rozwiązanie równania Poissona w obszarze jednowymiarowym	60
8.1.3. Dyskretyzacja dwuwymiarowego laplasjanu	62
8.1.4. Rozwiązanie równania Poissona w obszarze dwuwymiarowym	64
8.1.5. Dyskretyzacja i rozwiązanie równania Poissona w obszarze trójwymiarowym	67

8.2. Równanie dyfuzji . . . . .	71
8.2.1. Dyskretyzacja zadania jednowymiarowego . . . . .	72
8.2.2. Rozwiązanie równania dyfuzji w obszarze jednowymiarowym . . . . .	74
8.2.3. Dyskretyzacja dwuwymiarowego operatora dyfuzji . . . . .	75
8.2.4. Rozwiązanie równania dyfuzji w obszarze dwuwymiarowym . . . . .	77
8.3. Uwagi i uzupełnienia . . . . .	78
<b>9. Równanie logistyczne z opóźnieniem . . . . .</b>	<b>79</b>
9.1. Sformułowanie zadania i implementacja . . . . .	79
9.2. Weryfikacja wyników . . . . .	82
<b>10. Zadanie telekomunikacyjne . . . . .</b>	<b>84</b>
10.1. Konstrukcja macierzy przejścia . . . . .	86
10.1.1. Implementacja w małej skali: Octave . . . . .	87
10.2. Wyznaczenie stanu stacjonarnego . . . . .	90
10.2.1. Wybór właściwej metody . . . . .	91
10.3. Zadanie w dużej skali . . . . .	93
10.3.1. Implementacja procedury generowania macierzy w C . . . . .	93
10.3.2. Rozwiązanie zadania w Octave . . . . .	94
<b>11. Równanie reakcji–dyfuzji . . . . .</b>	<b>96</b>
11.1. Rozwiązanie numeryczne równania zdyskretyzowanego w przestrzeni . . . . .	98
11.2. Wizualizacja wyników . . . . .	105
11.2.1. Wizualizacja w OpenDX . . . . .	105
11.2.2. Wizualizacja w ParaView . . . . .	105
11.3. Weryfikacja wyników . . . . .	105
<b>12. Czy te symulacje mogą kłamać? . . . . .</b>	<b>108</b>
12.1. Pierwsze błędne symulacje . . . . .	109
12.2. Kłopotliwe pytania . . . . .	111
12.3. Ponure konstatacje . . . . .	113
12.4. Światelko w tunelu . . . . .	115
<b>Literatura . . . . .</b>	<b>117</b>

# Wprowadzenie

Celem przedmiotu *Obliczenia Naukowe* jest zapoznanie uczestników zajęć z praktycznymi aspektami wykorzystania metod numerycznych i narzędzi programistycznych do rozwiązywania zagadnień matematyki stosowanej.

Na kolejnych zajęciach będziemy rozważać konkretne zadania obliczeniowe, jakie pojawiają się w realnych zastosowaniach: w elektrotechnice, biologii, telekomunikacji, chemii, rolnictwie, astrofizyce, itd. Jest tradycją tego przedmiotu, że co roku pojawiają się w nim nieco inne zadania, dlatego omówione tutaj przykłady należy traktować jako pewien zestaw, hmmm... *przykładowy*.

Należy podkreślić, że nasza praca — w odróżnieniu od typowej roli matematyka stosowanego — będzie zdecydowanie łatwiejsza niż w rzeczywistości i ograniczy się wyłącznie do rozwiązania na komputerze *dobrze zdefiniowanego problemu matematycznego* i przedstawienia wyników. W prawdziwym życiu, jest to tylko część pracy matematyka: wcześniej musi on pomóc poprawnie sformułować model matematyczny badanego zjawiska (co zdecydowanie nie jest trywialne i zazwyczaj wymaga wiedzy wykraczającej poza samą matematykę), następnie go przeanalizować<sup>1</sup> na gruncie teorii (słynne pytania egzystencjalne), co może w dalszym ciągu spowodować konieczność wprowadzenia modyfikacji do modelu. Symulacje numeryczne — czyli to, czym będziemy się zajmować podczas zajęć — mogą (i powinny!) stanowić kolejny etap falsyfikacji modelu i być inspiracją do jego ewentualnej zmiany. Są oczywiście również niezbędne wtedy, gdy zechcemy mieć jakikolwiek wgląd w *ilościowe* cech rozwiązań modelu.

Większość z prezentowanych tu zadań jest na tyle nieskomplikowana, że dają się one w pełni rozwiązać w ramach jednej pary zajęć: wykład + ćwiczenia w laboratorium. To też zwykle jest dalekie od rzeczywistości i dlatego przedstawimy tu także kilka przykładowych zadań o nieco większej złożoności, których rozwiązanie może wymagać kilku(?) godzin naszej pracy (wszelako dla komputera będą one wciąż „łatwe” i ich finalne rozwiązanie zajmie mu co najwyżej kilkadziesiąt sekund!).

Na zajęciach będziemy zakładać wstępną podstawową znajomość Octave lub MATLABa (zwykle z tymi systemami można się zetknąć na zajęciach laboratoryjnych z *Matematyki Obliczeniowej I*) oraz języka C (wykładanego w ramach *Wstępu do Informatyki*), jednak w razie potrzeby niezbędne wstępne wiadomości zostaną przedstawione na bieżąco podczas zajęć. Przyda się także podstawowa umiejętność pracy w systemie operacyjnym Linux. Naszym podstawowym narzędziem pracy będzie Octave, z języka C będziemy korzystać sporadycznie.

Jeśli odczuwasz pewien niepokój, czy Twoje umiejętności programistyczne/komputerowe są wystarczające, przejrzyj kilka początkowych rozdziałów niniejszego skryptu. Zajrzyj zwłaszcza do rozdziału [Octave: jak w nim pracować](#): zobaczysz, jak łatwa i wygodna jest praca z systemem Octave. Podczas zajęć będziemy oczywiście, jeśli tylko zajdzie taka potrzeba, dokładnie omawiać używane konkretne narzędzia lub konstrukcje programistyczne. Możesz też zawsze skorzystać ze [ściągawki](#) do Octave lub dokumentacji [online](#) do [Octave](#) i do [MATLABa](#) (oba są *prawie* identyczne). Jeśli już znasz system operacyjny Windows, praca w Linuxie wyda Ci się zaskakująco łatwa.

---

<sup>1</sup> Niestety, nie zawsze to jest możliwe i wtedy zostają już tylko symulacje numeryczne...

**Literatura** Będziemy dużo programować, więc będziesz potrzebować dokumentacji, o której była mowa powyżej. Podręcznik [20] może Ci służyć zarówno jako przewodnik po obliczeniach naukowych, jak i jako źródło wiadomości uzupełniających i poszerzających materiał omawiany na wykładach i zajęciach w laboratorium. Nieco podobna w duchu jest praca zbiorowa [7]. Podstawowy programowania w MATLABie są omawiane w wielu książkach obecnych na rynku, a także w internecie [26]; niezastąpionym bazowym podręcznikiem języka C jest [17].

# 1. Potrzeba i ból obliczeń numerycznych

Matematyka komputerowa z biegiem czasu stała się nie tylko fetyszem nowoczesności, ale także realną koniecznością — zwłaszcza w matematyce stosowanej. Osoba z nowoczesnym wykształceniem matematycznym powinna więc umieć skorzystać z pakietu komputerowego wspomagającego warsztat matematyka. Z drugiej strony, powinna także być świadoma nie tylko *możliwości*, ale także **ograniczeń** używanych narzędzi.

W niniejszym wykładzie zrobimy dość ogólny przegląd dostępnych narzędzi oraz potencjalnych problemów związanych ze stosowaniem komputerowego wspomagania podczas uprawiania matematyki.

Dlatego naszym celem podczas wykładu będzie

- osiągnięcie **sprawności** w korzystaniu z systemu obliczeń naukowych i umiejętności wyszukiwania jego nowych funkcji, potrzebnych do rozwiązania nowego problemu;
- wyrobienie w sobie głębokiego **krytycyzmu** odnośnie uzyskanych wyników; nawyku ich **weryfikacji** metodami komputerowymi i pozakomputerowymi.

W życiu codziennym tak często korzystamy z komputera, że już niemal w pełni uzależniliśmy się od tej technologii. Przecież nie wyobrażamy sobie życia bez elektronicznej obsługi w banku i sklepie, bez komputerowego wyszukiwania informacji; z trudem przychodzi nam napisać list bez pomocy edytora tekstu. Komputery od swych początków znalazły sobie także poczesne miejsce w naukach ścisłych, ale głównie jako bardzo sprawne, wielkie kalkulatory. Od lat 70. ubiegłego stulecia komputery torują sobie uparcie drogę także w stronę zastosowań w dziedzinach nauki dotychczas zastrzeżonych dla ludzi, związanych z szeroko rozumianym myśleniem. Także w matematyce.

Przyzwyczajeni do niezawodności komputerów (któż z nas po każdej płatności kartą sprawdziłby nerwowo zgodność stanu konta?) zapominamy, że — jak każdy wytwór człowieka — komputery są też niedoskonałe. Oswoił się z myślą, że albo komputer działa dobrze, albo źle, a jak już działa źle, to w efekcie zawiesza się. Tymczasem sytuacja jest o wiele bardziej skomplikowana.

Zwłaszcza w matematyce, jak w żadnej innej dziedzinie wspomaganej komputerem, możemy paść ofiarą zbytnej ufności w możliwości maszyny. Może okazać się, że wynik działania komputera będzie *wyglądać sensownie*, ale w istocie będzie *całkowicie fałszywy* — a komputer nie da nam najmniejszego znaku, że coś poszło nie tak, jak powinno. Wykorzystanie komputera w matematyce najczęściej nie zwalnia od posiadania gruntownej wiedzy matematycznej: komputer będzie jedynie przystawką do intelektu użytkownika, a specjalistyczne oprogramowanie — okazją do sprawdzenia jego kultury matematycznej i przytomności umysłu.

W takim razie, po co używać tych narzędzi, skoro nie można im bezgranicznie ufać? Odpowiedź jest taka sama, jak na pytanie, po co robić eksperymenty w naukach przyrodniczych, skoro zawsze obarczone są błędem pomiarowym: po prostu nic lepszego nie mamy i dlatego musimy na wiele sposobów zabezpieczyć się przed zafałszowaniem wyników. Zatem obliczenia naukowe są tylko jeszcze jednym (ułomnym jak wszystko, co ludzkie) sposobem **eksperymentowania**, a nie — źródłem prawdy absolutnej.

## 1.1. Przegląd metod i narzędzi

### 1.1.1. Rachunek symboliczny kontra obliczenia numeryczne

#### Rachunek symboliczny

Umiejętność znalezienia rozwiązań abstrakcyjnego zadania matematycznego może dać nam wgląd we właściwości modelowanych zjawisk. Mało tego, wyrażając rozwiązanie w zależności od *parametrów zadania*, za pomocą *wzoru*, możemy wprost stwierdzić, jak ich zmiana wpływa na zachowanie się rozwiązania. Dlatego taką wagę przykładamy w matematyce do *rachunku symbolicznego*, gdzie operacje prowadzi się na abstrakcyjnych symbolach. Wydawać by się więc mogło, że komputerowe systemy obliczeń symbolicznych (*Computer Algebra Systems*, CAS) byłyby idealnym narzędziem w pracy matematyka stosowanego. Prowadząc rachunki symboliczne, dokonujemy *matematycznych* przekształceń pewnych wyrażeń zawierających abstrakcyjne symbole matematyczne. Są to więc w naturalny sposób rachunki *dokładne* ...o ile oczywiście nie pomylimy się w trakcie przekształceń! Niestety, takie pomyłki zdarzają się nie tylko ludziom, ale także najlepszym pakietom komputerowym CAS.

#### Obliczenia numeryczne

Z drugiej strony, czasem nie tyle chodzi nam o *wzór na rozwiązanie ogólnego problemu*, ale raczej o *wartość liczbową konkretnego rozwiązania konkretnego problemu*. Wynik numeryczny — czyli konkretna liczba w postaci dziesiętnej, z ustaloną liczbą wyznaczonych cyfr znaczących — jest wynikiem *przybliżonym*. Mało tego, on *z natury* jest wynikiem przybliżonym, bo

- dane potrzebne do jego uzyskania (np. eksperymentalnie wyznaczana „stała” przyspieszenia ziemskiego  $g \approx 9.81$ ) są znane jedynie w przybliżeniu
- obliczenia wykonywane są przez jednostkę arytmetyczną procesora, która wykonuje działania matematyczne z ograniczoną precyzją (tzw. arytmetyka zmiennoprzecinkowa).

Wydaje się więc, że w porównaniu z nieskończoną dokładnością wyników rachunków symbolicznych, obliczenia numeryczne są mniej wartościowe. Jednak, co zaskakujące, tak wcale nie jest! Korzystając z obliczeń numerycznych, można odpowiedzieć na pytanie o przybliżoną wartość rozwiązania konkretnego zadania także wtedy, gdy rachunek symboliczny zawodzi<sup>1</sup> — na przykład, kiedy *nie daje się wyznaczyć jawnego wzoru na rozwiązanie*.

#### Porównanie

Spróbujmy podsumować podobieństwa i różnice rachunków: symbolicznego i numerycznego. Bezsprzecznie, *rachunek symboliczny jest kwintesencją matematyki czystej*. Przeprowadzenie go od ogólnie sformułowanego zadania z parametrami do końcowego, eleganckiego rozwiązania (na przykład: wzoru) wymaga zazwyczaj dużej wiedzy matematycznej, finezji, szczęścia. Bywają jednak zadania, w których nie jesteśmy w stanie podać precyzyjnej formuły na rozwiązanie — na przykład nie można wyznaczyć funkcji pierwotnej dla  $e^{-x^2}$  w terminach funkcji elementarnych.

O ile więc rachunek symboliczny może być (*nomen omen*) symbolem matematyki czystej, o tyle *rachunek numeryczny jest koniem pociągowym matematyki stosowanej*. Modele matematyczne i inne zadania, z jakimi spotykają się w życiu zawodowym osoby korzystające z matematyki jako narzędzia pracy (inżynierowie, analitycy bankowi, biolodzy, fizycy, ekonomiści, ...) są najczęściej na tyle skomplikowane, że badanie ich metodami matematyki teoretycznej często jest niemożliwe, bardzo trudne, albo nieopłacalne. Natomiast metody numeryczne bardzo dobrze sobie radzą z takimi zadaniami.

Tak więc, dwa rachunki — symboliczny i numeryczny — *uzupełniają się*.

<sup>1</sup> Na niektóre pytania o *jakościowe* cechy rozwiązań zadań matematycznych można również odpowiedzieć na gruncie czystej teorii.



- W idealnym przypadku na wszystkie pytania dotyczące zadanego modelu matematycznego powinniśmy udzielić wyczerpujących odpowiedzi, opierając się wyłącznie na gruncie teorii. Jednak w wielu praktycznych zadaniach jest to bardzo trudne, gdyż może być zbyt czasochłonne lub najzwyczajniej wykraczać poza nasze możliwości. Tam, gdzie finezja matematyka–teoretyka nie jest w stanie sobie poradzić z zadaniem, znakomitym ratunkiem jest wykorzystanie rachunku numerycznego. Najczęściej tak jest, gdy problem ma bardzo wiele parametrów, które nie mają sztywno ustalonych wartości, albo gdy po prostu nie sposób go rozwiązać podając wzór na rozwiązanie, albo gdy akurat nie mamy pod ręką dostatecznie sprawnego matematyka, który w dodatku ma czas na poświęcenie się naszemu problemowi. Matematykę obliczeniową bezsprzecznie najlepiej uprawia się z pomocą szybkiego komputera i często brutalna siła tysięcy (milionów?) obliczeń — wykonywanych przez nieraz całkiem wyrafinowane algorytmy — prowadzi do oczekiwanego przez nas rezultatu. Musimy jednak pamiętać, że nawet w wyjątkowo trudnych przypadkach może być wciąż dodatkowo potrzebna ludzka wiedza matematyczna, zręczność programistyczna i pomysłowość.
- Tam, gdzie rachunki numeryczne zawodzą (na przykład, gdy mamy do czynienia z szeroko pojętymi nieskończonościami, albo gdy zadanie jest zbyt abstrakcyjne, albo gdy dostępna precyzja obliczeń jest zbyt mała do prawidłowego przeprowadzenia symulacji) — tam trzeba wrócić na trudną drogę i korzystać z metod matematyki czystej, „teoretycznej”. Czasem może wspomóc ją oprogramowanie potrafiące przeprowadzić żmudne (te nudne!) rachunki symboliczne, jednak i te wyliczenia powinien sprawdzić człowiek: systemy CAS są wciąż mocno niedoskonałe.

Komputerowy rachunek numeryczny ma znacznie dłuższą historię, przez co wydaje się znacznie dojrzalszy od komputerowych narzędzi do obliczeń symbolicznych. Z drugiej jednak strony, spektrum zadań, które mogą być stawiane przed systemami algebry komputerowej jest znacznie szersze od klas zadań ulegających metodom numerycznym. Jasne jest, że ideałem byłby system hybrydowy, wybierający — w zależności od konkretnego zadania — albo rachunek symboliczny, albo numeryczny, albo jakąś ich miksturę, w zależności od tego, co najlepiej zadziała w danej sytuacji. Niestety, żaden istniejący system nie posiadał w pożądanym stopniu tej cechy. Co więcej, nawet potencjalnie najdoskonalszy „system” — człowiek — również często może mieć z tym kłopoty...

Podsumujmy:

Metoda rachunku	numeryczny	symboliczny
Możliwość rozwiązywania trudnych zadań praktycznych	zazwyczaj tak	zazwyczaj nie
Wielość metod o różnej skuteczności	tak	tak
Wymaga wiedzy wykraczającej poza rozwiązywane zadanie	najczęściej tak	najczęściej nie
Wynik	skończony zestaw liczb lub rysunek	wzór lub informacja o charakterze rozwiązania
Potrafi działać na abstrakcyjnych obiektach	nie	tak
Dobrze radzi sobie z nieskończonościami	zazwyczaj nie	zazwyczaj tak
Dobrze radzi sobie z mnogością parametrów	tak	nie
Precyzja wyniku	ograniczona	teoretycznie nieskończona
Ostateczna jakość wyniku	niepewna	niepewna

Ostatni wiersz tabeli może być szokujący.

Jak to?! Prowadząc rachunek symboliczny można nie być pewnym ostatecznego wyniku?!!

Tak właśnie jest, niezależnie od tego, czy rachunek prowadzi maszyna, czy człowiek.

Z kolei rachunek numeryczny ze swej natury jest obarczony niepewnością dlatego, że wszystkie wykonywane obliczenia mają ograniczoną (choć całkiem dużą) dokładność. Co gorsza, okazuje się, że nawet proste obliczenia numeryczne mogą prowadzić do wielce niedokładnych wyników... Na szczęście ten fakt jest powszechnie znany i praktycznie każdy podręcznik metod numerycznych zawiera informacje na ten temat, zob. np. serwis <http://wazniak.mimuw.edu.pl> i literaturę tam cytowaną.

Co więc wybrać? Jakie podejście? Niestety, nie ma prostej odpowiedzi. Decyzja wymaga zazwyczaj pewnej wiedzy, doświadczenia, intuicji. A te cechy można rozwinąć tylko przez samodzielną pracę.

### 1.1.2. Środowiska komputerowe wspomagające pracę matematyczną

W świetle tego, co dotychczas zostało powiedziane, wydawać by się mogło, że matematyczne pakiety komputerowe będą dzielić się na pakiety numeryczne i pakiety symboliczne — i popularnie rzeczywiście takie rozróżnienie się przyjmuje. W rzeczywistości jednak, oprogramowanie matematyczne dzieli się na następujące dwie grupy:

**Pakiety numeryczne** Są wyspecjalizowane w prowadzeniu szybkich obliczeń numerycznych w dużej skali i mają także bogate możliwości graficznej wizualizacji wyników. Zwykle polecenia dostępne w pakietach numerycznych przewyższają pod względem skuteczności obliczeń ich analogony z pakietów wielozadaniowych.

**Pakiety wielozadaniowe** Przede wszystkim koncentrują się na obliczeniach symbolicznych, ale również udostępniają funkcje numeryczne i wizualizacyjne, a nawet dają możliwości składu tekstów matematycznych. Z biegiem lat, liczba i jakość funkcji obliczeń numerycznych w tych pakietach zwiększa się, bo — jak już stwierdziliśmy w rozdziale 1.1.1 — te dwie formy prowadzenia rachunków matematycznych uzupełniają się. Ze względu na nacisk na obliczenia symboliczne, czasem w skrócie, dla odróżnienia od pakietów czysto numerycznych, pakiety wielozadaniowe nazywane są pakietami obliczeń symbolicznych, bądź systemami algebry komputerowej (CAS), czy też wręcz po prostu pakietami matematyki komputerowej..

Najprawdopodobniej przyszłość należy do zintegrowanych środowisk wielozadaniowych. Jednak bieżąca praktyka wciąż uczy, że na razie lepiej korzystać z kilku wyspecjalizowanych pakietów, niż z jednego wielozadaniowego. Wciąż podstawowym narzędziem komputerowym matematyki stosowanej są (czasem bardzo złożone) obliczenia numeryczne.

Wszystkie popularne pakiety matematyczne (numeryczne i symboliczne) są konstruowane w sposób niezależny od platformy sprzętowej i dlatego dostępne dla wszystkich popularnych systemów operacyjnych komputerów osobistych, takich jak: Windows, Linux, czy MacOS, a także dla superkomputerów.

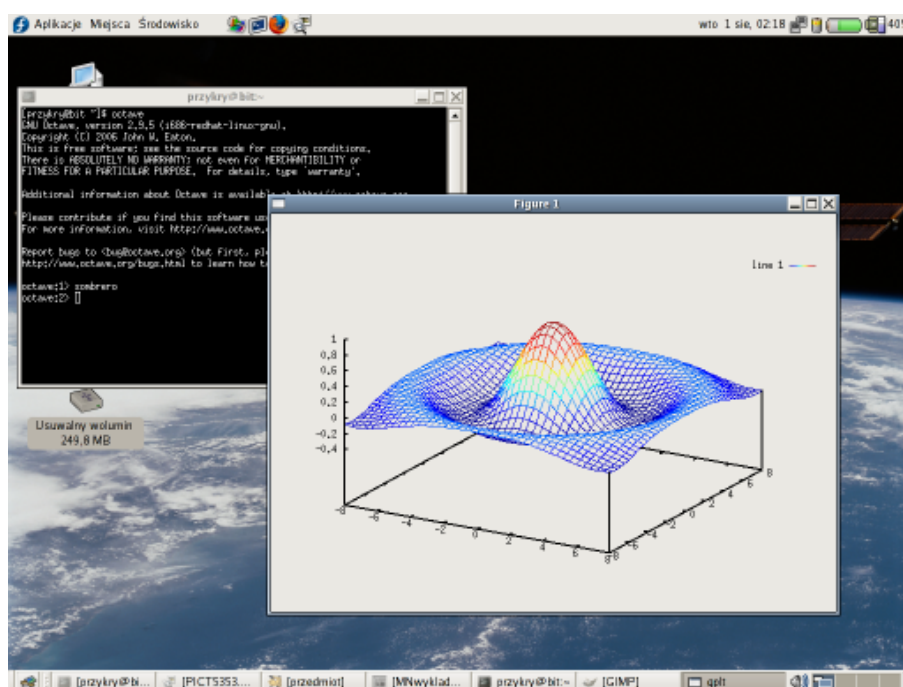
Poniżej przedstawiamy przegląd wybranych programów komputerowych przydatnych w szeroko rozumianych obliczeniach naukowych.

#### Pakiety numeryczne

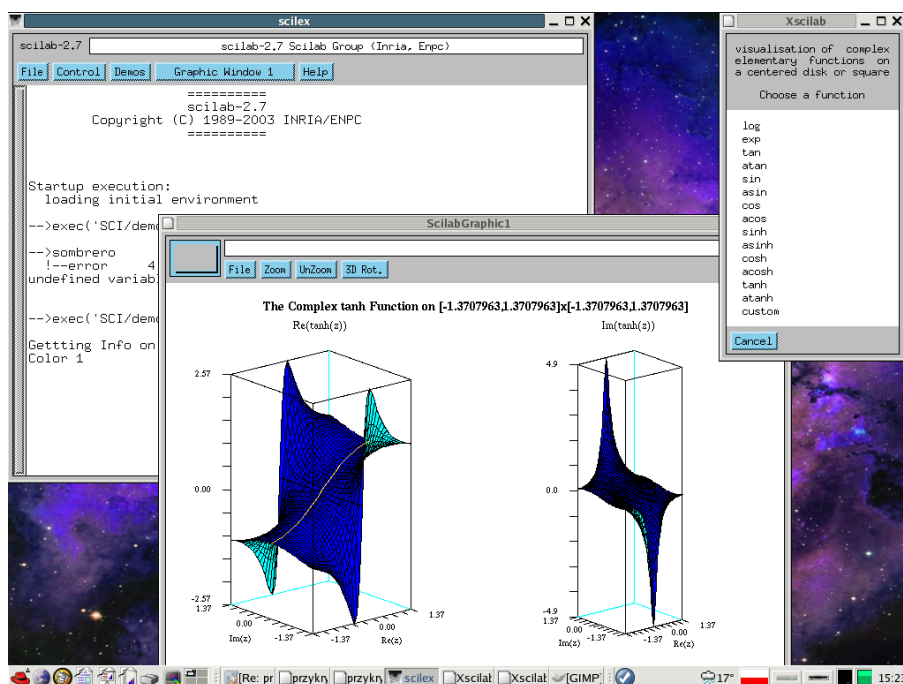
**MATLAB** <http://www.mathworks.com> Komercyjny.

W końcu lat siedemdziesiątych ubiegłego wieku, Cleve Moler wpadł na pomysł stworzenia prostego interfejsu do ówczesnie istniejących bibliotek numerycznych algebry liniowej. Stworzone przez niego: język skryptów i łatwe w użyciu narzędzia manipulacji macierzami, zdobyły wielką popularność w środowisku naukowym. W 1984 roku Moler skomercjalizował swe dzieło pod nazwą MATLAB (od: „Matrix Laboratory”).





Rysunek 1.2. Sesja Octave. W terminalu otwarta sesja w ascetycznym trybie tekstowym, grafika wyświetlana z wykorzystaniem Gnuplota.



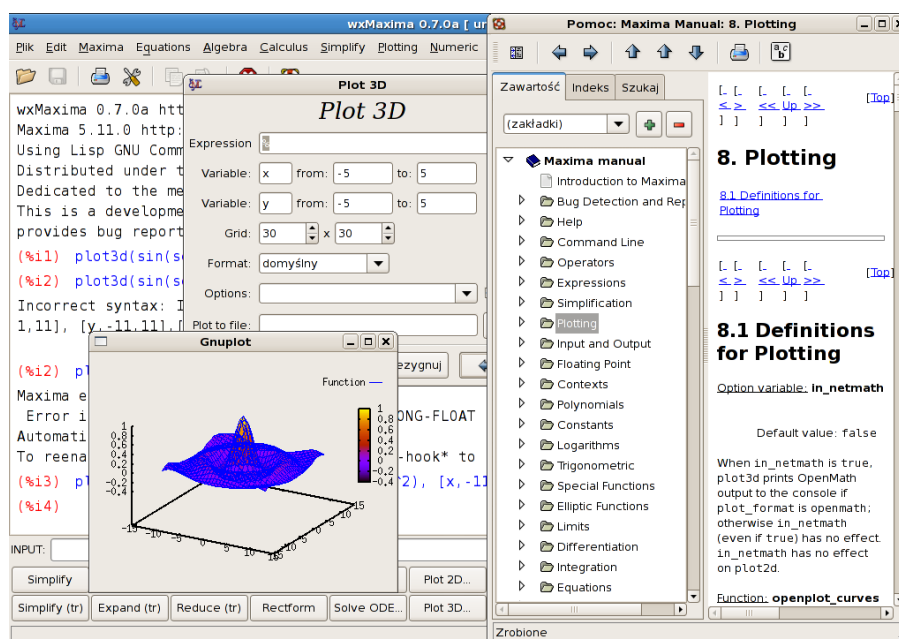
Rysunek 1.3. Sesja Scilaba..

Macsyma<sup>2</sup>, opracowanego na początku lat siedemdziesiątych ubiegłego stulecia w Massachusetts Institute of Technology (MIT) w USA. System zaprogramowano w języku Lisp i implementowano na kilku dostępnych w tamtym czasie komputerach typu *mainframe*. Był to pierwszy poważ-

<sup>2</sup> Nazwa pochodzi od angielskiego *Project MAC's Symbolic Manipulator*.

ny system obliczeń symbolicznych i jego koncepcja wpłynęła w wielkim stopniu na późniejsze systemy, takie jak Mathematica, czy Maple (ich składnia jest w dużym stopniu wzorowana na Macsymie). Lata 80. to etap burzliwy w dziejach Macsymy: z jednej strony, pojawiają się pierwsze implementacje na maszynach Unix’owych, z drugiej strony, jest podejmowanych kilka nieudanych prób komercjalizacji tego systemu, co w efekcie doprowadziło do stagnacji rozwoju i pełnej utraty rynku<sup>3</sup>. W tym czasie pojawiło się wiele gałęzi rodowych tego systemu o podobnie brzmiących nazwach; jedną z nich była tzw. DOE Macysma, rozwijana przez prof. [Williama Scheltera](#) od 1982 roku. W 1998 roku otrzymał on zezwolenie na pełne upublicznienie tej wersji kodu i od tamtej pory DOE Macysma trafiła do puli wolnego oprogramowania (na tzw. licencji GPL), pod obecną nazwą Maxima. W. Schelter zmarł nagle w 2001 roku, lecz dzięki temu, że projekt zdomowił się wcześniej w społeczności twórców wolnego oprogramowania — *software* przetrwał i rozwija się aż po dzień dzisiejszy.

Maxima jest systemem starym, z jednej strony z tradycjami, ale z drugiej strony wyraźnie niedoinwestowanym w ostatnich latach; jak trafnie punktuje konkurencja, m.in. brak w niej wszystkich usprawnień wprowadzanych do komercyjnych wersji Macsymy przez całe lata 80. Brak także szybkich procedur numerycznych (które miała wersja komercyjna, korzystająca z profesjonalnej biblioteki numerycznej IMSL, a potem także z LAPACKa). Jej komercyjni rywale odnieśli druzgocące zwycięstwo na rynku, pozostawiając *open-source*’owej Maximie niszę w postaci zastosowań w tych obszarach, gdzie zakup (skądinąd niebywale drogich) licencji na wiodące oprogramowanie komercyjne jest niemożliwy lub ekonomicznie nieuzasadniony. Jest ona także bardzo wdzięcznym polem do popisu dla osób pragnących wspomóc swoim intelektem i pracą społeczność twórców i użytkowników wolnego oprogramowania, a w ten sposób — zdobyć bardzo cenne doświadczenie.



Rysunek 1.4. Sesja Maximy.

System Maxima jest przede wszystkim zorientowany na rachunki symboliczne, choć możliwe są w nim także obliczenia numeryczne (i to w dowolnej precyzji, przez co wykonują się bardzo wolno). Interfejs użytkownika jest bardzo przyjemny, w mojej ocenie nawet lepszy od systemów

<sup>3</sup> M.in. zupełnie zaniedbano błyskawicznie rozwijający się segment komputerów osobistych.

komercyjnych. Wybrane procedury mają także swoje numeryczne odpowiedniki<sup>4</sup> — lecz zwykle o innej nazwie, co jest mniej eleganckie w porównaniu np. z konsekwentnym rozwiązaniem zastosowanym w Mathematicie. Maxima ma kilka znanych słabości (m.in. rozwiązywanie bardzo ważnego w praktyce zagadnienia upraszczania wyników pozostawia bardzo wiele do życzenia; także numeryka poważnie kuleje), jednak wciąż jest to system zupełnie dobry dla okazjonalnych użytkowników.

**Maple** <http://www.maplesoft.com> Komercyjny.

Środowisko obliczeń symbolicznych stworzone<sup>5</sup> w roku 1981 przez Symbolic Computation Group na Uniwersytecie Waterloo (Kanada), a następnie w roku 1988 skomercjalizowane. Obecnie jest to bardzo zaawansowane środowisko obliczeń symbolicznych. Ma także bogate możliwości wizualizacyjne oraz coraz lepsze funkcje numeryczne (w części numerycznej opiera się na bardzo dobrych bibliotekach NAG). Okazjonalnemu użytkownikowi zapewne spodoba się rozbudowany system pomocy, z licznymi przykładami zastosowania każdej funkcji systemu.

**Mathematica** <http://www.wolfram-research.com> Komercyjny.

Zaawansowane, a przy tym niesłychanie popularne środowisko — głównie obliczeń symbolicznych. Ma także bogate możliwości numeryczne i graficzne. Jego niewątpliwą zaletą jest bardzo spójny, konsekwentny sposób wydawania poleceń (choć z początku trzeba się z nim chwilę oswoić). Mathematica została stworzona w roku 1988 przez Stephena Wolframa i jego współpracowników, od początku z myślą o użytkownikach komputerów osobistych — najpierw Macintosh, lecz szybko doczekała się implementacji na wszystkie wówczas popularne systemy komputerów osobistych i stacji roboczych (MS Windows, SunOS, NeXT). Bardzo popularna na uniwersytetach, także za sprawą znakomitych podręczników oraz elegancji składni, preferującej styl programowania funkcyjnego.

**Axiom** <http://wiki.axiom-developer.org> Darmowy.

Bardzo powikłane są losy ciekawego skądinąd systemu Axiom. Jego prekursorem był Scratchpad firmy IBM, rozwijany od początku lat 70. ubiegłego stulecia. Po zmianie strategii IBM na początku lat 90. został on zakupiony przez potentata oprogramowania dla obliczeń numerycznych, NAG Ltd, i wówczas zmienił nazwę na obecną. Niestety, system nie odniósł komercyjnego sukcesu i z tego powodu NAG zdecydowała się w 2001 roku udostępnić program i jego kody źródłowe (niestety z pewnymi wyjątkami, dotyczącymi Aldora, kompilatora bibliotek Axioma) jako wolne oprogramowanie. W ten sposób Axiom tylko częściowo trafił do społeczności wolnego oprogramowania i m.in. dlatego wciąż nie zyskał sobie popularności. Jednak wszystko wskazuje na to, że już wkrótce całość Axiom/Aldor będzie udostępniona jako *open source*.

### Pakiety uzupełniające

Nie należy zapominać o pakietach, które także oferują możliwości prowadzenia wielu obliczeń matematycznych, zarówno na poziomie codziennej użyteczności metod matematycznych, jak i na poziomie wyrafinowanych zastosowań profesjonalnych. Poniżej wymieniamy kilka najpopularniejszych i ich cechy charakterystyczne.

**Excel** <http://www.microsoft.com/excel> Komercyjny.

<sup>4</sup> Niestety, część z nich nie może korzystać z możliwości zwiększenia precyzji obliczeń.

<sup>5</sup> Był to odruch desperacji wobec faktu, że w owym czasie system Macsyma wymagał bardzo drogiego sprzętu komputerowego — w rzeczywistości firma posiadająca prawa dystrybucji Macsymy uważała ją za jeden ze sposobów zwiększenia sprzedaży własnych (dużych) komputerów.



Znakomity arkusz kalkulacyjny (składnik pakietu biurowego [Microsoft Office](#)), dający narzędzia prowadzenia, na danych liczbowych, prostych obliczeń matematycznych (także statystycznych) oraz ich wizualizacji niezbędnych w życiu codziennym. Należy wszak pamiętać, że arkusze kalkulacyjne (Excel i OpenOffice Calc) starają się ukryć przed użytkownikiem to, że precyzja ich obliczeń jest ograniczona, przez co tym groźniejsze mogą być czynione przez nie błędy zaokrągleń!<sup>6</sup>

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	1,1	1,1	1,1	1,1	1,1	1,1	1,1	1,1	1,1	1,1	1,1	1,1	1,1	1,1	1,1	1,2	2,09	10,97	99,79
2	Różnica	0,00E+000	0	0	0	0	0	0	0	0	0	0	0	0	-0,01	-0,09	-0,89	-8,88	-88,82

Rysunek 1.5. Excel/Calc nie prowadzi dokładnych rachunków na ułamkach dziesiętnych, chociaż czasem udaje, że tak. (Na obrazku: OpenOffice Calc).

Niektórzy są do tego stopnia zachwyceni możliwościami obliczeniowymi arkusza kalkulacyjnego, że propagują nawet wykonywanie w nim zaawansowanych obliczeń naukowych [9]. My jednak stoimy na stanowisku, że do konkretnych zadań należy dobierać właściwe narzędzia i z tej pozycji *odradzamy* stosowanie w obliczeniach naukowych Excela i mu podobnych: z pewnością można skuteczniej je prowadzić w środowiskach takich jak Octave.

**OpenOffice Calc** <http://www.openoffice.org/calc>

Darmowy odpowiednik Excela o równie potężnych możliwościach. Składnik pakietu biurowego [OpenOffice.org](#), gruntownie omawiany w ramach kursu [Przysposobienia informatycznego](#).

**S i S-plus** [www.insightful.com](http://www.insightful.com) Komercyjny.

Wielce popularny i bardzo mocny program do prowadzenia profesjonalnych analiz statystycznych.

**R** <http://www.r-project.org>

Darmowy i znakomity odpowiednik S. *De facto standard* analizy statystycznej. O stopniu jego dojrzałości niechaj świadczy perfekcyjna dokumentacja oraz znaczna liczba publikacji naukowych wykorzystujących ten pakiet do analizy danych.

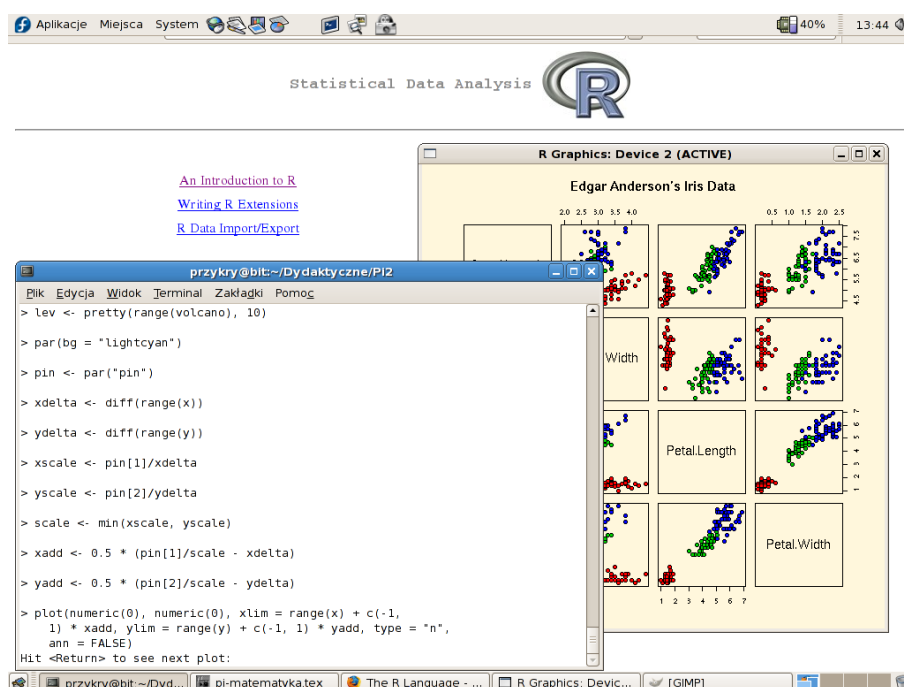
**SAS** <http://www.sas.com> Komercyjny.

Bardzo zaawansowane narzędzie analizy (głównie statystycznej) wielkich zbiorów danych, używane m.in. w bankowości.

**AVS** <http://www.avs.com> Komercyjny.

Narzędzie profesjonalnej wizualizacji danych naukowych.

<sup>6</sup> Spróbuj np. w komórce A1 arkusza wpisać liczbę 1,1, i do komórki obok wpisać formułę  $=10*(A1-1)+0,1$ . Następnie przeciągnij za uchwyt wypełniania i skopiuj tę formułę do kolejnych dwudziestu komórek na prawo tak, że np. w komórce L1 będzie formuła  $=10*(K1-1)+0,1$ . Gdyby obliczenia wykonywały się w nieskończonej precyzji, w każdej z komórek dostałbyś ten sam wynik 1.1. I faktycznie, arkusz przekonuje Cię, że w pierwszych paru komórkach tak jest faktycznie (jeśli chcesz, to sprawdź, że  $=B1-A1$  daje jakoby *dokładnie zero* (nie zapomnij nadać wynikowi „formatu naukowego” — Excel domyślnie wyświetla wyniki z dokładnością do dwóch cyfr po przecinku). Jednak, jak możesz przekonać się na własne oczy, wcale tak nie jest! (Przykład zaczerpnięty z wykładu C. Wolframa)



Rysunek 1.6. Pakiet R w akcji.

## OpenDX <http://www.opendx.org>

Darmowy, lecz wysokiej jakości odpowiednik AVS, bezpośredni spadkobierca komercyjnego **Data Explorera**, którego całość kodu źródłowego producent, IBM, upublicznił i dalej wspiera jego rozwój na zasadach wolnego oprogramowania.

System może być używany zarówno jako niezależne środowisko do wizualizacji danych wczytywanych w odpowiednim formacie z pliku, a także jako biblioteka wywoływana z poziomu innego programu. Typowy przebieg pracy z OpenDX, podobnie jak z każdym innym systemem wizualizacji danych, ma cztery etapy:

1. wybór danych do wizualizacji, przygotowanie ich w formacie strawnym dla OpenDX;
2. wczytanie danych do OpenDX;
3. wstępna obróbka, np. wybór skali kolorów, rodzaju wizualizacji, itp.
4. obrazowanie i interpretacja danych oraz dalsza manipulacja obrazem (wydobywanie ukrytych informacji) — a więc w rzeczywistości „**goto 3**”.

Pierwszy etap pracy przebiega na zewnątrz OpenDX — jest zadaniem aplikacji generującej dane podlegające późniejszej wizualizacji. To aplikacja prowadząca symulację numeryczną musi zapisać wyniki w jednym z formatów strawnych dla OpenDX. Drugi etap, czyli wczytanie danych do OpenDX, w przypadku standardowych danych jest całkiem prosty, gdyż OpenDX dysponuje narzędziem wspomagającym import danych, tzw. Data Prompterem.

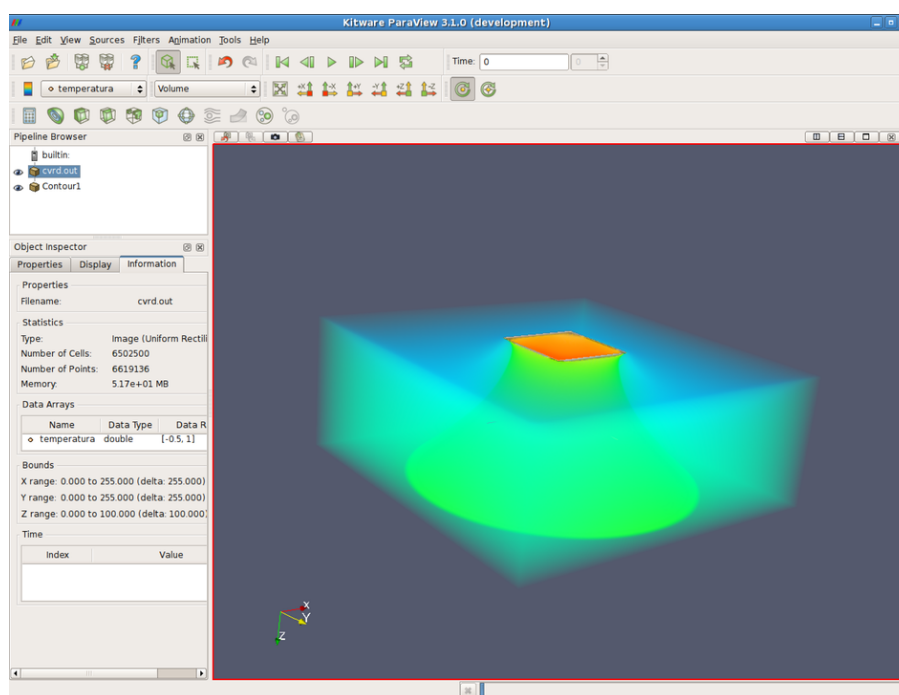
Kluczem do dobrej wizualizacji jest krok trzeci, czyli wstępna obróbka danych przy użyciu narzędzi udostępnianych przez OpenDX. Obróbka odbywa się w sposób dosyć spektakularny, mianowicie poprzez łączenie ze sobą podstawowych modułów takich jak: nadawanie koloru, definiowanie przekrojów, wyznaczanie gradientów, itp. Takie połączone ze sobą moduły tworzą wspólnie *sieć wizualizacyjną*. Praca z OpenDX polega głównie na utworzeniu i kolejnych modyfikacjach sieci wizualizacyjnej.

Krok czwarty to wykorzystanie możliwości interakcyjnych OpenDX oraz dalsza modyfikacja sieci wizualizacyjnej dla osiągnięcia zamierzonego efektu.



## ParaView <http://www.paraview.org>

Jednym z alternatywnych dla OpenDX narzędzi wizualizacji dużych zbiorów danych naukowych jest ParaView [23] — wspólny projekt Kitware oraz amerykańskich rządowych laboratoriów Sandia i Livermore. System ten jest oparty na uznanej i wciąż rozwijanej bibliotece procedur wizualizacyjnych VTK (*Visualization ToolKit*) [31] — i na swój sposób stanowi wygodny, okienkowy interfejs do tej biblioteki. Podobnie jak OpenDX, ParaView daje możliwość wczytywania danych w rozmaitych zewnętrznych formatach, ale głównie wspiera formaty opracowane dla biblioteki VTK. Zasadniczym mankamentem ParaView jest to, że jego dokumentacja [27], w przeciwieństwie do samego programu, nie jest darmowa.



Rysunek 1.7. Wizualizacja trójwymiarowego pola skalarnego w ParaView.

## Podstawowe zasady bezpieczeństwa i higieny pracy z pakietami matematycznymi

- Nigdy nie wierz na ślepo wynikom, ani tym bardziej ich wizualizacji. *Użycie komputera nie zwalnia od myślenia.*
- Gdy tylko to możliwe, korzystaj z opcji uzyskania informacji<sup>7</sup> o zastosowanej metodzie, o szacowanym błędzie wyniku, o przyczynach nie rozwiązania zadania.
- Pakiety mogą dawać błędną odpowiedź nawet wtedy, gdy wykonywane są jedynie rachunki symboliczne.
- Dokładny wzór na rozwiązanie nie zawsze jest najlepszym rozwiązaniem zadania.
- Niewielkie zaburzenie parametrów zadania może spowodować kolosalne zmiany rozwiązania.
- Szybciej nie zawsze znaczy: lepiej.
- Ludzka ingerencja w rachunki symboliczne prowadzone na bieżąco na komputerze, zazwyczaj prowadzi do znacznego zwiększenia elegancji końcowych wyników.

<sup>7</sup> Z reguły, łatwiej takie informacje otrzymać w pakietach numerycznych; pakiety symboliczne najczęściej starają się działać na zasadzie „czarnej skrzynki”.

- Nie warto uruchamiać programów obliczeniowych metodą prób i błędów, „aż się skompiluje”: to dobra metoda na wprowadzenie do kodu subtelnych i bardzo trudnych do wykrycia pomyłek.
- Większość zadań dotyczących teoretycznych właściwości modelu nie daje się rozwiązać za pomocą symbolicznych rachunków na komputerze. *Jednak są duże szanse, że Tobie uda się je zbadać bez pomocy komputera*, dzięki doświadczeniu i specjalistycznej wiedzy.
- Ładny rysunek nie musi być wykresem prawdziwego rozwiązania.

## 2. Octave: jak w nim pracować

Poniżej przedstawiamy nagranie dwóch krótkich sesji w Octave, pokazujących podstawy wygodnej pracy z tym systemem.



Zobacz animację: *Podstawy pracy z Octave*, znajdującą się na stronie WWW przedmiotu. Pokazano m.in., jak uruchamiać Octave, wykonywać proste obliczenia, interpretować wyniki i kończyć pracę.

W bieżącej pracy z systemem Octave może przydać się [ściągawka do Octave](#), zawierająca na dwóch kartkach formatu A4 zestawienie najczęściej używanych poleceń. Warto ją wydrukować, zgiąć na trzy części i mieć pod ręką podczas pracy.



Zobacz animację: *Skrypty i funkcje w Octave*, znajdującą się na stronie WWW przedmiotu. Pokazano m.in., w jaki sposób można tworzyć skrypty i funkcje w Octave.

Znacznie więcej na temat używania i programowania w Octave można znaleźć w podręczniku obliczeń naukowych i inżynierskich [20]. Ze względu na bardzo wysoki stopień zgodności z MATLABem, do nauki języka programowania Octave'a można również wykorzystać popularne podręczniki, takie jak dostępne na stronach WWW wprowadzenie do (jednej ze starszych wersji) MATLABa [26]. Warto wszakże pamiętać, że w niektórych kwestiach MATLAB i Octave różnią się, i to nawet istotnie. Pełna dokumentacja Octave'a jest dostępna na stronach WWW projektu, [4].

### 3. Funkcja odwrotna

W analizie pewnego modelu dotyczącego transportu morfogenów w tkankach [19] istotną rolę odgrywa funkcja

$$\Sigma(s) = \frac{2}{M-1} (1 - (1-s)^{M-1}) - s(1-s)^{M-1}, \quad s \in [0, 1),$$

a ściślej: funkcja odwrotna do niej,  $\Sigma^{-1}$ . Naszym zadaniem jest naszkicowanie wykresu  $\Sigma^{-1}$  dla różnych wartości  $0 \leq M < 1$  i zbadanie, jak bardzo  $\Sigma^{-1}(z)$  jest bliskie 1, gdy  $z$  jest rzędu kilkudziesięciu.

Jasne jest, że wszelkie nasze komputerowe działania musimy zacząć od zdefiniowania funkcji  $\Sigma$ . Zrobimy to w Octave:

```
function v = sigmafun(s,M)
v = (2/(M-1))*(1-(1-s).^(M-1)) - s.*(1-s).^(M-1);
end
```

(Dalsze rozważania będziemy prowadzić dla  $M = 1/2$ ). Jeśli chcieć tylko z grubsza naszkicować wykres odwrotnej, to zadanie robi się banalne: wykres  $\Sigma^{-1}$  to zbiór punktów postaci

$$(z, \Sigma^{-1}(z)), \quad z \in [0, \infty),$$

a to jest przecież „to samo”, co zbiór

$$(\Sigma(s), s), \quad s \in [0, 1).$$

Wobec tego wystarczy skorzystać z funkcji **plot**, ale „na opak”:

```
M = 0.5;
s = linspace(0,1);
plot(sigmafun(s,M), s, 'r-', 'linewidth', 2);
```

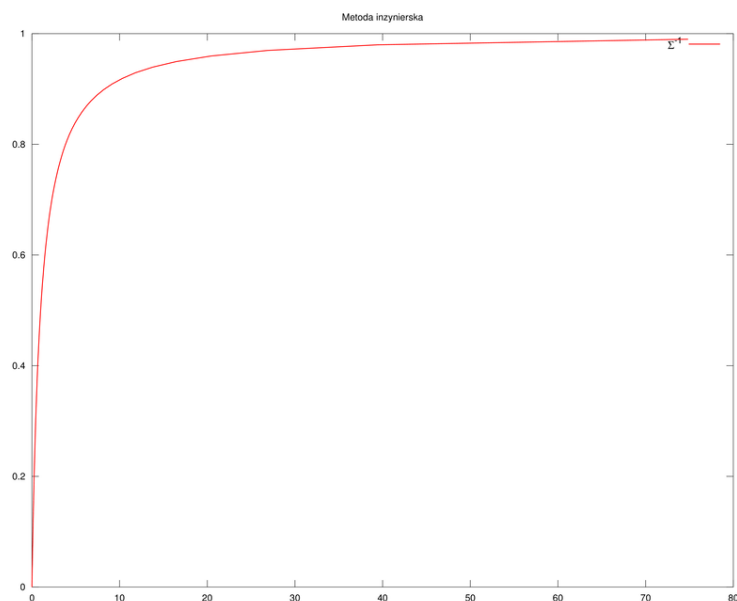
Z wykresu na rysunku 3.1 widzimy, że  $\Sigma^{-1}$  bardzo szybko przybliża się do wartości granicznej równej 1, co sugerowałoby wybór innej skali dla wizualizacji: tym, co mogłoby nas *naprawdę* zainteresować, mogłaby być szybkość przybliżania się  $\Sigma^{-1}$  do 1: wybralibyśmy więc wówczas, zamiast zwykłego **plot**, znacznie bardziej wyraziste **semilogy**(sigmafun(s,M), 1-s). Jednak dla ustalenia uwagi, w dalszym ciągu pozostaniemy przy wykresie w skali liniowej.

#### 3.1. Doprecyzowanie zadania

Niepokojący jest prawy koniec wykresu: wszak  $\Sigma(1)$  jest nieokreślone! Zatem doprecyzujmy nasze zadanie: w naszym konkretnym zastosowaniu, chcielibyśmy wiedzieć, ze względu na naturę badanego zjawiska, jak blisko 1 jest  $\Sigma^{-1}(60)$  i narysować wykres  $\Sigma^{-1}$  na odcinku  $[0, 60]$ .

Napiszemy zatem w Octave funkcję, która będzie wyznaczać wartości funkcji odwrotnej do  $\Sigma$  dla dowolnego *zadanego*  $z \in [0, \infty)$ . Idea jest następująca: z definicji funkcji odwrotnej,  $\Sigma^{-1}(z) = S$ , gdzie  $S$  spełnia  $\Sigma(S) = z$ , a więc  $S$  jest miejscem zerowym funkcji

$$F_z(S) = \Sigma(S) - z.$$



Rysunek 3.1. Wykres funkcji odwrotnej do  $\Sigma$ , uzyskany metodą „inżynierską” dla  $M = 1/2$ . Ze względu na bardzo szybki wzrost  $\Sigma$  (i nieokreśloność  $\Sigma(1)$ ), wykres dostajemy jedynie dla  $z < 26$ .

Aby dla zadanego  $z$  znaleźć  $S$ , wystarczy znaleźć miejsce zerowe skalarnej funkcji  $F_z$  — a to już powinno być łatwe, gdy skorzystamy z funkcji **fzero**. Powyższą ideę implementuje funkcja **sigmainv** z poniższego listingu.

```
function [S, fc] = sigmainv(z, M)
if z == 0
    S = 0;
    fc = 0;
else
    [S, FS, info, out] = fzero(@(s)Fz(s,z,M), [0,1-eps/2]);
    if(info ~= 1)
        warning(['Kłopoty dla z=', num2str(z), '; info=', num2str(info)]);
    end
    fc = out.funcCount; % zliczamy liczbę wywołań Fz
end
end % sigmainv

function Z = Fz(S, z, M) % "z" jest parametrem
Z = sigmafun(S, M)-z;
end
```

Zanim przyjrzymy się meritum funkcji **sigmainv**, zwróćmy uwagę na kilka programistycznych szczegółów. Jak pamiętamy, **fzero** żąda funkcji jednego argumentu, dlatego, dla każdego zadanego  $z$ , konstruujemy taką funkcję *ad hoc* — na podstawie trzyargumentowej  $F_z(S,z,M)$  — korzystając z mechanizmu funkcji anonimowej: funkcja

```
f = @(s)Fz(s,z,M)
```

jest funkcją *jednej* zmiennej, z wartościami  $z$  i  $M$  takimi, jakie były w chwili jej definiowania. Samą funkcję  $F_z$  określiliśmy zaś jako funkcję lokalną dla `sigmainv`. Wreszcie, aby uniknąć konsternacji **fzero** dla  $z = 0$  (czyim spowodowanej?) zwracamy zawsze  $\Sigma^{-1}(0) = 0$ , bez niepotrzebnych obliczeń.

Jest oczywiste, że  $\Sigma(s)^{-1}$  jest monotoniczna i może przyjmować wartości z przedziału  $[0, 1)$ . Dlatego wiedząc, że dla  $s = 1$  funkcja  $\Sigma(s)$  jest nieokreślona, przedział lokalizujący miejsce zerowe ustaliliśmy na  $[0, 1-\text{eps}/2]$ , gdyż  $1-\text{eps}/2$  jest największą liczbą maszynową mniejszą od 1 (dlaczego?).

Pozornie „asekurancki” sposób zdefiniowania funkcji `sigmainv`, w której *zawsze* sprawdzamy wartość parametru **info** zwracaną przez **fzero**, jest w istocie bardzo ważną i sensowną decyzją, pozwalającą wstępnie zweryfikować wyniki. Wszak nie wiemy *a priori*, czy nasz *solver* sobie poradzi z  $F_z$ !

### 3.2. Transformacja zadania do wygodniejszej postaci

Chociaż wykres dostajemy bez trudu, stosując prostą *metodę kontynuacji* — biorąc za przybliżenie początkowe wartość z poprzedniego punktu wykresu (por. wykład z [Matematyki Obliczeniowej II](#)), to przychodzi nam nań czekać dość długo. Gdy ponownie spojrzymy na wykres  $\Sigma$ , sprawa staje się jasna: przecież dla  $s \approx 1$ ,  $\Sigma$  ma *piekielnie* stromy wykres, a to znacznie utrudnia działanie *solvera* powodując, że praktycznie ogranicza się on do metody bisekcji. Aby uniknąć tej niedogodności i znacząco przyspieszyć rysowanie wykresu, skorzystamy ze starej, dobrej zasady prowadzenia obliczeń numerycznych (i nie tylko):

Gdy zadanie jest za trudne, należy zmienić... zadanie!

Tutaj po prostu zastąpimy „trudną” funkcję  $F_z$  inną, „łatwiejszą”, ale o identycznych miejscach zerowych. Prosty rachunek pokazuje nam, że  $s$  jest rozwiązaniem równania  $\Sigma(s) - z = 0$  wtedy i tylko wtedy, gdy  $s$  jest miejscem zerowym funkcji

$$G_z(s) \equiv 1 - s - \left( \frac{p-s}{p+z} \right)^{p/2}, \text{ gdzie } p = 2/(1-M).$$

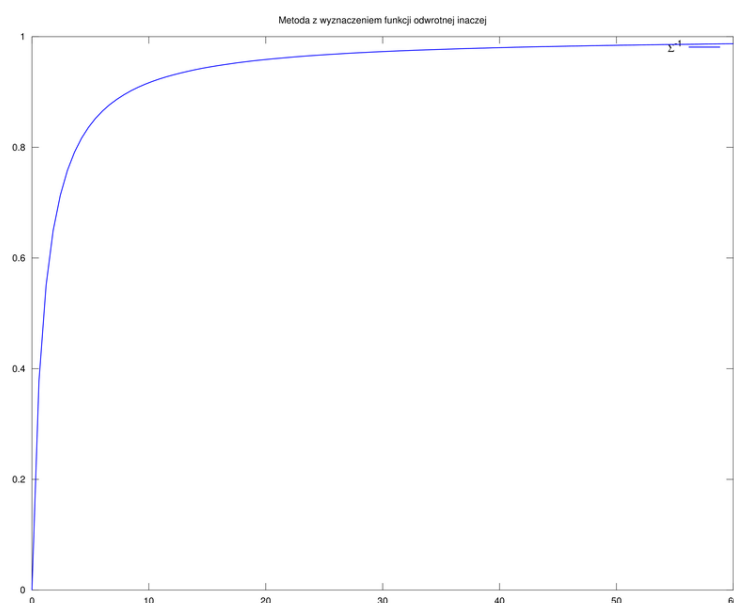
Różnica między  $F_z$  a  $G_z$  jest taka, że ta ostatnia jest prawie liniowa, więc rzeczywiście powinna być „łatwa” dla *solvera* równania nieliniowego. Przy okazji,  $G_z$  nie ma już osobliwości: jest określona nawet dla  $s \geq 1$ .

Wyznaczanie wartości  $\Sigma^{-1}(z)$  przez znalezienie miejsca zerowego funkcji  $G_z$  pokazuje szczegółowo kolejny listing.

```
function [S, fc] = sigmainv2(z,M)
if z == 0
    S = 0;
    fc = 0;
else
    [S, FS, info, out] = fzero(@(s)Gz(s,z,M), [0,1]);
    if(info ~= 1)
        warning(['Kłopoty_dla_z=', num2str(z), ', info=', num2str(info)]);
    end
    fc = out.funcCount; % zliczamy liczbę wywołań Gz
end
end % sigmainv2
```

```
function Z = Gz(S, z, M) % "z" jest parametrem
p = 2/(1-M);
Z = 1 - S - ((p - S)./(p+z)).^(p/2);
end
```

Faktycznie, dzięki tej zmianie wykres  $\Sigma^{-1}$  (rysunek 3.2) dostajemy trzy razy szybciej aniżeli w przypadku poprzedniego podejścia, przy liczbie wywołań funkcji  $G_z$  sześciokrotnie mniejszej niż w przypadku  $F_z$ .



Rysunek 3.2. Wykres funkcji odwrotnej do  $\Sigma$ , uzyskany metodą przez znalezienie miejsca zerowego funkcji  $G_z$ .

### 3.3. A może wystarczy po prostu... wziąć większy młotek?

Na zakończenie warto być może zauważyć, że w dzisiejszych czasach obliczenia są na tyle tanie, że nie zawsze warto odwoływać się do aż tak wyrafinowanych metod. Gdyby naszym głównym zadaniem było narysowanie, wyłącznie w celach poglądowych, jednego wykresu na odcinku  $[0, 60]$ , to zapewne byłoby nam dość obojętne, czy wyznaczymy sto, sto tysięcy, czy nawet milion wartości `sigmafun`: nasz komputer najpewniej i tak jest wystarczająco szybki, by z tym sobie błyskawicznie poradzić. Komenda `logspace(k,m,N)` pozwala wygenerować zestaw  $N$  węzłów „równoodległych w skali logarytmicznej”, czyli postaci  $x_i = 10^{p_i}$ , gdzie  $p_i$  tworzą zestaw  $N$  równoodległych punktów z przedziału  $[k, m]$ .

Generując więc zestaw punktów skupiających się wokół  $s = 1$ , a następnie wybierając tylko te, dla których  $\Sigma(s) \leq 60$ :

```
N = 10000;
s = 1-logspace(-13,0,N); % punkty zagęszczają się wykładniczo wokół 1
Z = sigmafun(s,M);
good = find(Z<=60); % odrzucamy te, które wykraczają poza zakres zmiennej "z"
```

```
plot(Z(good), s(good));
```

udałoby się nam w ten sposób (nawet jeszcze szybciej, niż poprzednimi metodami!) „dociągnąć” wykres  $\Sigma^{-1}$  do  $z = 59.999$ , co jest chyba całkiem niezłym wynikiem, jak na metodę brutalnej siły...

(Ale i tutaj ważne jest wyczucie: gdyby nieopatrznie położyć  $s = 1 - \text{logspace}(-300, 0, N)$ , dostalibyśmy znacznie *gorszy* wynik).

**Ćwiczenie 3.1.** Czasami funkcja może być na tyle złośliwa, że zamiast zmieniać jest postać — jak to zrobiliśmy powyżej — możemy spróbować skorzystać z mniej wyrafinowanej, a za to ogólniejszej metody rozwiązywania równania nieliniowego  $F_z(S) = 0$ . Taką bardzo prostą metodą, o minimalnych wymaganiach wobec  $F_z$ , jest metoda bisekcji (zadowolili się funkcją ciągłą, zmieniającą znak). Zaimplementuj metodę bisekcji i zobacz, jak sprawdzi się w warunkach naszego problemu dla oryginalnej funkcji  $F_z$ .



## 4. Reakcja enzymatyczna

### 4.1. Równanie Michaelisa–Menten

W jednym z prostszych modeli reakcji enzymatycznych pojawia się równanie Michaelisa–Menten, ustanawiające zależność szybkość reakcji  $v_0$  od stężenia substratu  $S$ :

$$v_0 = \frac{V S}{K_m + S}. \quad (4.1)$$

Współczynniki  $K_m$  i  $V$  są pewnymi parametrami ( $K_m$  nazywa się stałą Michaelisa) — por. także wykład z [Modeli matematycznych w biologii i medycynie](#). Wartości  $v_0$  i  $S$  można zmierzyć doświadczalnie, [32] (zob. także [16]) podaje m.in. następujący zestaw danych:

$S$ [mmol dm <sup>-3</sup> ]	0.25	0.30	0.40	0.50	0.70	1.00	1.40	2.00
$v_0$ [μmol dm <sup>-3</sup> min <sup>-1</sup> ]	2.4	2.6	4.2	3.8	6.2	7.4	10.2	11.4

Naszym zadaniem jest wyznaczenie takich wartości parametrów  $K_m$  i  $V$ , które najlepiej pasowałyby do powyższych danych eksperymentalnych.

#### 4.1.1. Nietrafione uproszczenia

Wielu badaczy, jeszcze w czasach gdy obliczenia były zaporowo drogie, zaproponowało rozmaite transformacje powyższego problemu, pozwalające w ostatecznym rozrachunku sformułować zadanie jako *liniowe zadanie najmniejszych kwadratów*. Mając w pamięci numeryczną maksymę, że

Jeśli zadanie jest za trudne, należy... zmienić zadanie!

możemy ulec wrażeniu, że będzie to właściwa droga ku efektywnemu znalezieniu „optymalnych” wartości szukanych parametrów.

W przypadku zadania dopasowania  $K_m$  i  $V$ , używano (por. [11]) między innymi takich (matematycznie równoważnych) transformacji (4.1):

**Równanie czysto liniowe** Mnożąc (4.1) stronami przez  $K_m + S$  i dzieląc przez  $S$ , dostajemy równanie *liniowe* względem  $K_m$  i  $V$ :

$$K_m \frac{1}{S} - V \frac{1}{v_0} = -1, \quad (4.2)$$

które możemy oczywiście rozwiązać metodą najmniejszych kwadratów. Jeśli wektory kolumnowe  $S$  i  $v_0$  będą zawierać dane z pomiarów (odpowiednio:  $S_i$  i  $(v_0)_i$ ,  $i = 1, \dots, N$ ), to instrukcje

```
e = ones(size(S),1);
```

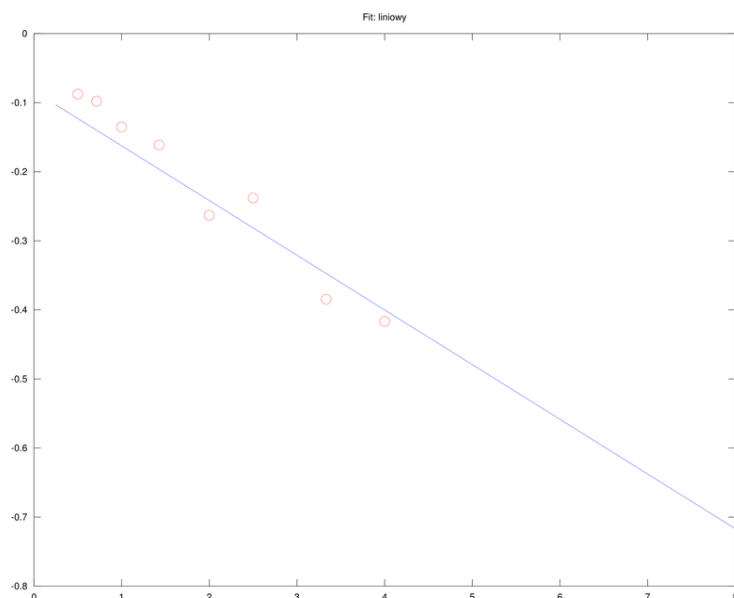
```
A = [1./S, -1./v0]; b = -e;
```

```
x = A \ b;
```

```
V = x(2); Km = x(1);
```

spowodują wyznaczenie  $K_m$  i  $V$  takich, które zminimalizują sumę

$$\sum_{i=1}^N \left( K_m \frac{1}{S_i} - V \frac{1}{(v_0)_i} + 1 \right)^2.$$



Rysunek 4.1. Dopasowanie prostej do danych w równaniu liniowym (4.2).

**Równanie Lineweavera i Burka** Jeśli poprzednio wyprowadzone równanie podzielimy przez  $V$ , dostaniemy równanie Lineweavera i Burka,

$$\frac{1}{v_0} = \frac{1}{V} + \frac{K_m}{V} \frac{1}{S} = x_2 + x_1 \frac{1}{S},$$

które jest liniowe względem pomocniczych zmiennych  $x_1 = \frac{K_m}{V}$ ,  $x_2 = \frac{1}{V}$ .

Analogicznie jak poprzednio, możemy znaleźć wartości  $x_1, x_2$ , które zminimalizują wyrażenie

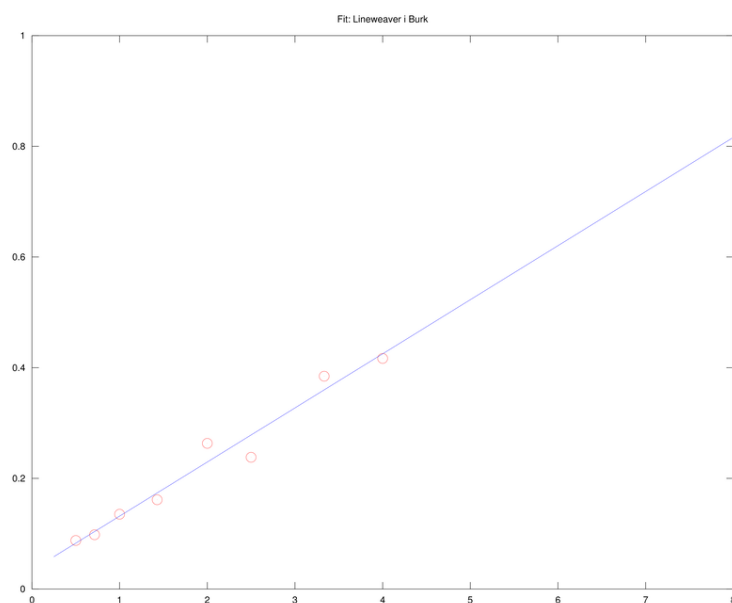
$$\sum_{i=1}^N \left( x_2 + x_1 \frac{1}{S_i} - \frac{1}{(v_0)_i} \right)^2.$$

Rozwiążemy je w Octave, korzystając z operatora „\”, sekwencją komend:

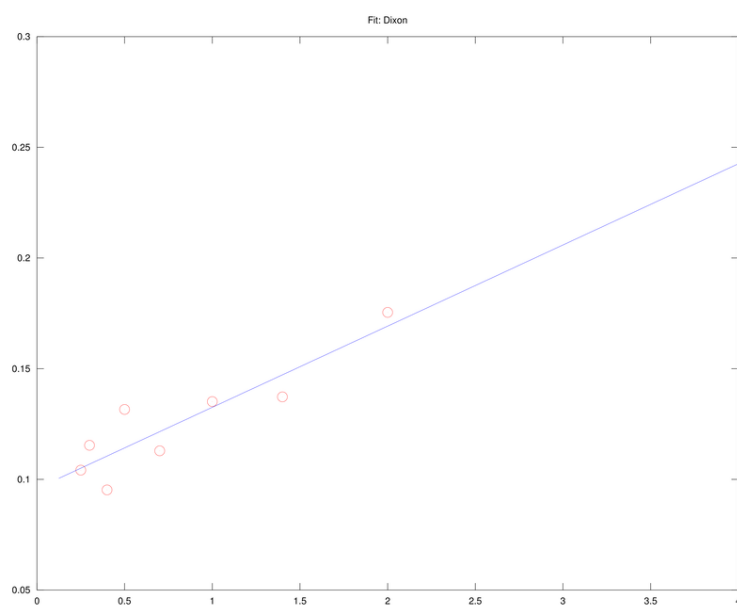
```
A = [1./S, e]; b = 1./v0;
x = A \ b;
V = 1/x(2); Km = x(1)*V;
```

**Równanie Dixona** Mnożąc równanie Lineweavera i Burka przez  $S$ , dostajemy równanie Dixona,

$$\frac{S}{v_0} = \frac{1}{V} S + \frac{K_m}{V} = x_2 S + x_1,$$



Rysunek 4.2. Dopasowanie prostej do danych w równaniu Lineweavera i Burka.



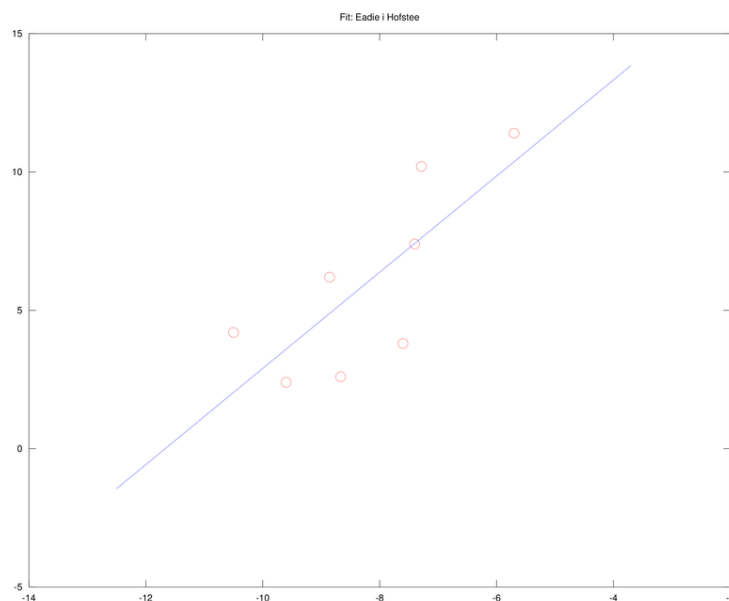
Rysunek 4.3. Dopasowanie prostej do danych w równaniu Dixona.

liniowe w pomocniczych zmiennych  $x_1 = \frac{K_m}{V}$ ,  $x_2 = \frac{1}{V}$ . Ponownie więc możemy dopasować  $K_m$  i  $V$ , rozwiązując liniowe zadanie najmniejszych kwadratów.

**Równanie Eadie i Hofstee** Mnożąc wreszcie równanie liniowe (4.2) przez  $v_0$ , dochodzimy do równania postaci:

$$-K_m \frac{v_0}{S} + V = v_0,$$

które jest liniowe w oryginalnych zmiennych,  $K_m$  i  $V$ . To zadanie też możemy rozwiązać przez liniowe zadanie najmniejszych kwadratów.



Rysunek 4.4. Dopasowanie prostej do danych w równaniu Eadie i Hofstee.

**Ćwiczenie 4.1.** Dla danych podanych w tabeli na początku rozdziału, wyznacz każdym z opisanych sposobów współczynniki  $K_m$  i  $V$ .

*Rozwiązanie.* Przykładowe rozwiązania znajdują się w listingu w dalszej części wykładu. Swoją odpowiedź możesz porównać z tabelką w następnym rozdziale.

### Krytyka metod prowadzących do liniowego zadania najmniejszych kwadratów

Powstaje więc — bynajmniej nie filozoficzne, ale czysto praktyczne — pytanie, która z metod jest „lepsza”, czyli która z nich da najlepsze możliwe dopasowanie (przy rozsądnym koszcie). Otóż wszystko zależy od tego, co będziemy rozumieli pod pojęciem „najlepsze”: wszak z definicji każde z uzyskanych przez nas rozwiązań było najlepsze, jako rozwiązanie zadania minimalizacji residuum postaci  $\|b - Ax\|_2^2$  dla zadanych  $A$  i  $b$ .

Ponieważ „naturalnym” sformułowaniem naszego zadania było (4.1), wydaje się równie naturalnym postawienie i ocena rozwiązania w sensie *nieliniowego* zadania najmniejszych kwadratów:

$$\phi(V, K_m) = \sum_{i=1}^N \left( (v_0)_i - \frac{V S_i}{K_m + S_i} \right)^2 = \min! \quad (4.3)$$

Jeśli porównać wartości  $\phi$  dla parametrów otrzymanych wyżej opisanymi metodami, to okaże się, że

równanie	$V$	$K_m$	$\phi(V, K_m)$	uwarunkowanie ( $\kappa$ )
Lineweaver i Burk	29.43	2.88	1.71	12.1
Dixon	27.27	2.61	1.58	7.13
Eadie i Hofstee	20.27	1.74	2.65	874
liniowe	12.00	0.95	23.40	3170

A więc w tym sensie, zdecydowanie najlepsze rezultaty dało dopasowanie metodą Dixona (por. także rysunek 4.5). Co więcej, okazuje się, że metody prowadzące do liniowej zależności od  $V$  i  $K_m$ , czyli równanie liniowe (4.2) i równanie Eadie i Hofstee są bardzo źle uwarunkowane<sup>1</sup>, ze współczynnikiem uwarunkowania  $\kappa$  rzędu  $10^3$ . Jest to o tyle ważne, że dane do naszego zadania zdają się być zmierzone z dokładnością względną  $\epsilon$  rzędu  $10^{-1} \dots 10^{-2}$ : taki błąd może więc skutkować zaburzeniem względnym wyniku na poziomie rzędu  $\kappa\epsilon$ , czyli na poziomie  $10^1 \dots 10^2$ .

Sensowne wyniki dają metody Lineweaver i Burk oraz Dixona, ale... czy nie można *jeszcze lepiej*? Wszak możemy spróbować rozwiązać *wprost* nieliniowe zadanie najmniejszych kwadratów (4.3), na przykład korzystając z *solvera* sqp, dostępnego w Octave.

#### 4.1.2. Rozwiązanie nieliniowego zadania najmniejszych kwadratów

*Solver* sqp jest narzędziem służącym do rozwiązywania znacznie bardziej złożonych zadań optymalizacji, dlatego jego użycie bywa skomplikowane. Ale w naszym prostym przypadku — minimalizacji bez ograniczeń, dla funkcjonału kwadratowego — nie będziemy musieli podawać mu zbyt wielu parametrów. Najpierw jednak zdefiniujemy funkcjonał, który będziemy minimalizować:

```
% (X(1), X(2)) <---> (V, K)
F = @(X,S) (X(1)*S)./(X(2)+S);
phi = @(X) sumsq(v0 - F(X,S));
```

Jak widać, zaczęliśmy od określenia funkcji

$$F_{V,K_m}(S) = \frac{V S}{K_m + S},$$

która pojawia się w definicji (4.1) i w (4.3), i która z pewnością nam się przyda: na przykład do narysowania wykresu  $F$  dla wyznaczonych  $V$  i  $K_m$  (dlatego definiujemy ją od razu wektorowo ze względu na  $S$ ). Potem, przy użyciu  $F$ , zdefiniowaliśmy  $\phi$ , przy czym — ponieważ  $\phi$  jest anonimową funkcją tylko jednego argumentu,  $X$ , to w jej definicji zostaną użyte wcześniej przez nas zdefiniowane wektory  $S$  i  $v0$ , zawierające dane zadania. Dzięki temu, łatwo nam będzie wyznaczyć sumę kwadratów współrzędnych wektora  $v0 - F(X,S)$  — do tego właśnie służy funkcja Octave `sumsq`.

Teraz wystarczy wywołać *solver* sqp na naszym funkcjonałe  $\phi$ , z początkowym przybliżeniem  $(V, K_m)^T$  wyznaczonym np. metodą Dixona:

```
% podane na wejściu sqp parametry (V, Km) zostały wyznaczone metodą Dixona
[x, phix, info, iter, nphi] = sqp([V;Km], phi)
V = x(1); Km = x(2);
```

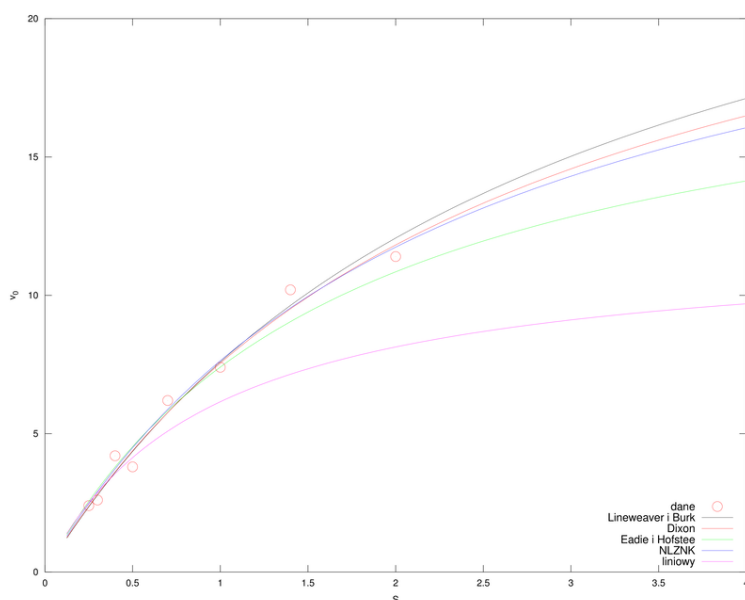
W wyniku dostajemy  $V = 25.40$  oraz  $K_m = 2.33$ , dla których  $\phi(V, K_m) = 1.51$ , a więc (nieco) lepiej niż dotychczas! Jak zwykle w przypadku metod iteracyjnych, zażądamy możliwie wielu informacji diagnostycznych, takich jak: kod zakończenia **info**, liczba wykonanych iteracji **iter**, oraz

<sup>1</sup> Współczynnik uwarunkowania zadania odzwierciedla podatność wyniku na zaburzenia danych. Gdy zadanie jest źle uwarunkowane (tzn. ma bardzo duży współczynnik uwarunkowania), nawet małe zaburzenie danych może spowodować bardzo duży błąd wyniku.

wywołań funkcjonalu `nphi`. Pożądana wartość **info** to, zgodnie z manuałem, 101: zakończenie z powodu nikłego postępu iteracji. Musimy jednak pamiętać, że znalezione minimum może być jedynie minimum lokalnym i wybór innego punktu startowego może dać w rezultacie znacznie lepszy (lub gorszy) wynik.

Podsumowując, w dzisiejszych czasach, gdy nawet dość zaawansowane obliczenia są (w miarę) łatwe i tanie, nie powinniśmy bać się nieliniowości. Pamiętajmy także, że jeśli tylko możemy sensownie wspomóc się przybliżeniem uzyskanym na drodze rozsądnej linearyzacji — warto z tej opcji skorzystać.

Na marginesie dodajmy, że opisane przez nas wcześniej metody sprowadzenia zadania do zadania liniowego, są wciąż rutynowo stosowane w innych zadaniach dopasowywania do punktów pomiarowych wykresu, na przykład postaci  $y(x) = ae^{bx}$ : biorąc logarytm od obu stron, ponownie dostajemy zadanie liniowe na  $a$  i  $b$ :  $\log(y) = \log(a) + bx$ . Jednak, jak już doświadczyliśmy, transformacja zadania najczęściej powoduje też zmianę sposobu, w jaki mierzymy błąd dopasowania. Niektóre transformacje dodatkowo mogą wyolbrzymiać lub redukować różnice pomiędzy danymi punktami pomiarowymi.



Rysunek 4.5. Dopasowanie krzywej (4.1) do danych dla różnych sposobów wyznaczenia parametrów.

```
S = [0.25 ; 0.30; 0.40; 0.50; 0.70; 1.00; 1.40; 2.00];
v0 = [2.4; 2.6; 4.2; 3.8; 6.2; 7.4; 10.2; 11.4];
```

```
F = @(X,S) (X(1)*S)./(X(2)+S);
fitfun = @(X) sumsq(v0 - F(X,S));
```

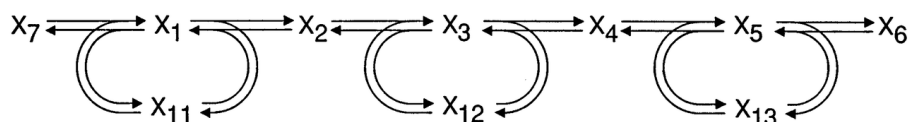
```
function cbA = lsqcond(A, b, x, tol)
if nargin < 4
    tol = 1e2;
end
```



To tylko fragment kodu źródłowego Octave. Więcej na <http://mst.mimuw.edu.pl/lecture.php?lecture=ona&part=Ch4>.

## 4.2. Różniczkowy model łańcucha reakcji enzymatycznych

W rozdziale 6. pracy [24] rozważa się model łańcucha trzech reakcji katalizowanych enzymatycznie. Niech  $X_7$  oznacza (dane) stężenie substratu,  $X_2$  i  $X_4$  — (niewiadome) stężenie dwóch produktów pośrednich, a  $X_6$  — (dane) stężenie produktu finalnego reakcji. Dalej, niech  $X_8, X_9, X_{10}$  będą zadanymi całkowitymi stężeniami trzech enzymów katalizujących każdą z reakcji (zob. rysunek 4.6 pochodzący z (Źródło: [24, rysunek 9].), przy czym  $X_1, X_3, X_5$  będą nieznanymi stężeniami odpowiednich enzymów związanych z produktami pośrednimi (a przez to nieczynnymi). Stężenia wolnych enzymów oznaczmy przez  $X_{11} = X_8 - X_1$ ,  $X_{12} = X_9 - X_3$ ,  $X_{13} = X_{10} - X_5$ .



Rysunek 4.6. Schematyczne przedstawienie omawianej reakcji enzymatycznej.

Wtedy kinetyka powyższego układu może być modelowana następującym zestawem równań różniczkowych zwyczajnych [24, równania (75)—(82)]<sup>2</sup>:

$$\begin{aligned}\frac{d}{dt}X_1 &= aX_7^cX_{11}^c + bX_2^cX_{11}^c - (b+a)X_1, \\ \frac{d}{dt}X_2 &= aX_1 + bX_3 - bX_2^cX_{11}^c - aX_2^cX_{12}^c, \\ \frac{d}{dt}X_3 &= aX_2^cX_{12}^c + bX_4^cX_{12}^c - (b+a)X_3, \\ \frac{d}{dt}X_4 &= aX_3 + bX_5 - bX_4^cX_{12}^c - aX_4^cX_{13}^c, \\ \frac{d}{dt}X_5 &= aX_4^cX_{13}^c + bX_6^cX_{13}^c - (b+a)X_5,\end{aligned}$$

z parametrami  $a, b, c$ , uzupełnionego warunkiem początkowym dla  $t = 0$ . Naszym zadaniem jest wyznaczenie wykresu zależności prędkości (netto)  $v$  powstawania produktu od czasu  $t$ , zgodnie ze wzorem [24, równanie (83)]

$$v(t) = aX_5 - bX_6^c(X_{10} - X_5)^c.$$

W tym celu w rutynowy sposób wyznaczmy rozwiązanie powyższego układu równań, korzystając z funkcji `lsode`. Najpierw jednak zapiszemy układ równań w formie takiej, by występowały w nim jedynie niewiadome  $X_1, \dots, X_5$  oraz zadane parametry  $X_6, \dots, X_{10}$ :

<sup>2</sup> Podajemy go już po pewnych uproszczeniach.

$$\frac{d}{dt} \begin{pmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \end{pmatrix} = \begin{pmatrix} aX_7^c(X_8 - X_1)^c + bX_2^c(X_8 - X_1)^c - (b+a)X_1 \\ aX_1 + bX_3 - bX_2^c(X_8 - X_1)^c - aX_2^c(X_9 - X_3)^c \\ aX_2^c(X_9 - X_3)^c + bX_4^c(X_9 - X_3)^c - (b+a)X_3 \\ aX_3 + bX_5 - bX_4^c(X_9 - X_3)^c - aX_4^c(X_{10} - X_5)^c \\ aX_4^c(X_{10} - X_5)^c + bX_6^c(X_{10} - X_5)^c - (b+a)X_5 \end{pmatrix}.$$

W naszych eksperymentach numerycznych przyjmiemy za [24]  $a = 2$ ,  $b = 1$ ,  $c = 4$ , a także  $X_8 = X_9 = X_{10} = 1$ . Ponadto  $X_7 = 1.425$  (badając model tak, jak w [24], sprawdzalibyśmy wpływ zmiany tego parametru na zmianę  $v$ ),  $X_6 = 1$ . W chwili  $t = 0$ , ustalamy  $X_1(0) = \dots = X_5(0) = 0$ . Oczywiście, cały skrypt symulujący przebieg reakcji zapiszemy w formie sparametryzowanej tak, by móc łatwo zmieniać wartości wszystkich danych zadania.

Funkcja prawej strony miałaby więc postać<sup>3</sup>

```
function dX = reaction(X,t)
global a;
global b;
global c;
global X6;
global X7;
global X8;
global X9;
global X10;

dX = [a*(X7*(X8-X(1)))^c + b*(X(2)*(X8-X(1)))^c - (b+a)*X(1) ;
a*X(1) + b*X(3) - b*(X(2)*(X8-X(1)))^c - a*(X(2)*(X9-X(3)))^c ;
a*(X(2)*(X9-X(3)))^c + b*(X(4)*(X9-X(3)))^c - (b+a)*X(3) ;
a*X(3) + b*X(5) - b*(X(4)*(X9-X(3)))^c - a*(X(4)*(X10-X(5)))^c ;
a*(X(4)*(X10-X(5)))^c + b*(X6*(X10-X(5)))^c - (b+a)*X(5)];
end
```

(dla większej skuteczności iloczynu postaci  $X^cY^c$  zapisaliśmy  $(XY)^c$ ). Natomiast skrypt symulacji mógłby wyglądać na przykład następująco:

```
global a;
global b;
global c;
global X6;
global X7;
global X8;
global X9;
global X10;

a = 2; b = 1; c = 4;
X6 = X8 = X9 = X10 = 1;
X7 = 1.425;
T = 10;

N = 800;
t = linspace(0,T,N);
X = lsode(@reaction, zeros(5,1), t);
```

<sup>3</sup> Jeżeli w definicji  $dX$  przypadkowo napiszesz  $dX = [a \ -b; \dots]$ , to interpreter Octave zrozumie, że chodzi o macierz o dwóch kolumnach,  $dX = [a, \ -b; \dots]$ . Dlatego, powinniśmy tutaj konsekwentnie pisać  $dX = [a \ -b; \dots]$  (z operatorem odejmowania otoczonym spacjami).



```
plot(t,X); pause(3);  
v = a*X(:,5) - b*(X6*(X10-X(:,5))).^c;  
plot(t,v)
```

Tradycyjnie, aby nieco upewnić się co do jakości otrzymanych wyników, powinniśmy przeprowadzić kilka symulacji z różnymi parametrami tolerancji błędu (i zbadać, czy przypadkiem nie padliśmy ofiarą *aliasingu*, czyli zbyt małej rozdzielczości wizualizacji, fałszującej rzeczywisty przebieg rozwiązania).

**Ćwiczenie 4.2.** Przeprowadź testy wizualnej i numerycznej jakości uzyskanego rozwiązania, zmieniając `N` (by zabezpieczyć się przed aliasingiem) oraz `lsode_options` (by zmniejszyć ryzyko wzięcia numerycznych artefaktów za prawdziwe własności rozwiązania).

## 5. Hodowla zwierząt

Aby ocenić wartość hodowlaną  $n$  zwierząt na podstawie  $m \leq n$  pomiarów wartości pewnej cechy, stosuje się pewien model liniowy, prowadzący do następującego zagadnienia, opisanego w [30, rozdział 5]:

$$\begin{pmatrix} X^T X & X^T Z \\ Z^T X & Z^T Z + kA^{-1} \end{pmatrix} \begin{pmatrix} b \\ a \end{pmatrix} = \begin{pmatrix} X^T y \\ Z^T y \end{pmatrix} \quad (5.1)$$

Szukane wartości hodowlane zwierząt oznaczone są jako wektor  $a \in \mathbb{R}^n$  (zatem  $i$ -te zwierzę ma wartość  $a_i$ ). Niewiadomymi pomocniczymi są parametry wpływu płciowości  $b \in \mathbb{R}^2$ .

Pozostałe parametry występujące w (5.1) —  $A$ ,  $X$ ,  $Z$ ,  $y$ ,  $k$  — są zadane jako parametry modelu. Jak to często się zdarza w przypadku obliczeń naukowych, będziemy dysponować jedynie częściową informacją o danych modelu, a naszym zadaniem będzie po prostu wskazanie sensownej metody numerycznego rozwiązywania układu równań liniowych (5.1). Co zatem na początek wiemy o parametrach modelu? Nasz „zleceniodawca” z pewnością zwróci nam uwagę na to, że macierze  $X$  oraz  $Z$  są macierzami zerojedynkowymi. Macierz  $X$  rozmiaru  $m \times 2$  określa płeć badanego zwierzęcia, a macierz  $Z$ , rozmiaru  $m \times n$ , odpowiada za „wkład” danego zwierzęcia do badanej cechy hodowlanej.

Wreszcie, o macierzy  $A$  rozmiaru  $n \times n$  — tzw. macierzy addytywnych pokrewieństw — wiadomo, że ma elementy nieujemne, jest symetryczna i dodatnio określona.

W praktyce [22, Table 1] spotyka się zadania dla  $n$  z zakresu od  $10^1$  do  $10^6$ . Macierz addytywnych pokrewieństw  $A$  w niektórych modelach może być pominięta (co odpowiada wartości parametru skalującego  $k = 0$ ), a w ogólności ze względu na swoją naturę powinna ona być dosyć rozrzedzona (hodowcy dążą do tego, by ograniczyć pokrewieństwa pomiędzy osobnikami).

Poniżej zacytujemy konkretne zadanie modelowe opisane w [30].

**Przykład 5.1** (Przykład 62 z [30]). W oparciu o metodę BLUP wykonać ocenę wartości hodowlanej zwierząt dla cechy *masa cieląt przy odsadzeniu* w oparciu o następujące informacje:

Ciele	Płeć	Ojciec	Matka	Waga przy odsadzeniu (kg)
4	buhajek	1	-	4,5
5	cieliczka	3	2	2,9
6	cieliczka	1	2	3,9
7	buhajek	4	5	3,5
8	buhajek	3	6	5,0

Zebrane informacje fenotypowe prowadzą do zadania (5.1), w którym

$$X = \begin{pmatrix} 1 & \\ & 1 \\ & 1 \\ 1 & \\ 1 & \end{pmatrix}_{5 \times 2}, \quad Z = \begin{pmatrix} & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & & 1 \\ & & & & & 1 \end{pmatrix}_{5 \times 8}, \quad y = \begin{pmatrix} 4.5 \\ 2.9 \\ 3.9 \\ 3.5 \\ 5.0 \end{pmatrix},$$

parametr  $k = 2$ , a symetryczna macierz addytywnych pokrewieństw jest zadana przez

$$A = \begin{pmatrix} 1 & & & & & & & \\ & 1 & & & & & & \\ & & 1 & & & & & \\ & & & 1 & & & & \\ sym & & & & 1 & & & \\ & & & & & 1 & & \\ & & & & & & 1 & \\ & & & & & & & 1 \end{pmatrix}_{8 \times 8}.$$

### 5.1. Dyskusja problemu

Zadanie wydaje się łatwe do rozwiązania: dane są macierze i parametry, wskazany jest układ  $n + 2$  równań do rozwiązania (5.1) — więc wystarczy zbudować jego macierz i go rozwiązać. Gdy zadanie nie jest zbyt wielkiego rozmiaru ( $n$  rzędu tysiąca?) — możemy je rozwiązać wprost w Octave. Rzeczywiście, [30, rozdział 10] podaje gotowy skrypt:

```
G = [X'*X, X'*Z ; Z'*X, Z'*Z + k*inv(A)];
r = [X'*y; Z'*y];
s = inv(G)*r;
b = s(1:2); a = s(3:end);
```

Przedyskutujmy wady i zalety powyższego rozwiązania. Tym, co od razu kłuje na w oczy, jest używanie funkcji **inv** do wyznaczania macierzy odwrotnej do  $G$  i do  $A$ . O ile macierz odwrotna do  $A$  występuje w samym sformułowaniu zadania, o tyle wyznaczenie rozwiązania układu równań

$$Gs = r$$

metodą  $s = \text{inv}(G)*r$  powinno zjeżyć nam włos na głowie. Oczywiście, choć matematycznie jest to poprawne, w realizacji numerycznej nie powinniśmy wyznaczać wprost macierzy odwrotnej! Znacznie lepiej dokonać rozkładu LDL macierzy  $G$  (wszak jest symetryczna!) i następnie rozwiązać dwa układy równań z macierzami trójkątnymi i jedną diagonalną. Ten algorytm realizuje w Octave operator „dzielenia” macierzowego:

```
s = G\r;
```

Zapatrzeni w odwracanie macierzy, możemy przeoczyć inną niepokojącą cechę układu (5.1): jeśli bowiem  $k = 0$ , to macierz naszego układu przyjmuje postać:

$$\begin{pmatrix} X^T X & X^T Z \\ Z^T X & Z^T Z \end{pmatrix}.$$

To jest przecież nic innego, jak macierz równań normalnych dla zadania najmniejszych kwadratów

$$\|y - (X \ Z) \begin{pmatrix} b \\ a \end{pmatrix}\|_2 \rightarrow \min!$$

— a zadanie najmniejszych kwadratów, jak wiemy, bezpieczniej rozwiązywać metodami innymi niż poprzez układ równań normalnych: na przykład, opartymi na rozkładzie QR macierzy  $\begin{pmatrix} X & Z \end{pmatrix}$ . Najpierw jednak musimy zadać sobie pytanie, czy również oryginalny układ (5.1) odpowiada jakiemuś zadaniu najmniejszych kwadratów? Możemy domyslać się, że tak (wiedząc o tym, jaki jest jego *rodowód*). I rzeczywiście, przecież dla dowolnej macierzy symetrycznej  $S$ ,

$$\begin{pmatrix} X & Z \\ 0 & S \end{pmatrix}^T \cdot \begin{pmatrix} X & Z \\ 0 & S \end{pmatrix} = \begin{pmatrix} X^T X & X^T Z \\ Z^T X & Z^T Z + S^2 \end{pmatrix}.$$

Biorąc więc  $S = k^{1/2}A^{-1/2}$  (istnieje, bo  $A$  jest symetryczna i dodatnio określona) dostajemy, że (5.1) jest układem równań normalnych dla zadania najmniejszych kwadratów:

$$\left\| \begin{pmatrix} y \\ 0 \end{pmatrix} - \begin{pmatrix} X & Z \\ 0 & S \end{pmatrix} \begin{pmatrix} b \\ a \end{pmatrix} \right\|_2 \rightarrow \min! \quad (5.2)$$

Problem z ostatnim sformułowaniem problemu polega na tym, że aby go postawić, musimy wyznaczyć  $A^{-1/2}$ , czyli — w rzeczywistości — rozwiązać pełne zagadnienie własne dla macierzy  $A$  (znaleźć wszystkie wektory i wartości własne).

Jednak po chwili namysłu możemy zobaczyć, że  $S$  nie musi być symetryczna, wystarczy tylko, żeby  $S^T S = kA^{-1}$ . Biorąc więc (łatwo obliczalny) rozkład Cholesky'ego–Banachiewicza macierzy  $A$ ,

$$A = LL^T,$$

dostajemy  $S = k^{1/2}L^{-1}$ . Ponieważ  $L$  jest macierzą dolną trójkątną, macierz  $L^{-1}$  można wyznaczyć przyzwoitym kosztem.

Ostatecznie więc, zamiast zadania (5.1) należy rozwiązać równoważne zadanie najmniejszych kwadratów,

$$\left\| \begin{pmatrix} y \\ 0 \end{pmatrix} - \begin{pmatrix} X & Z \\ 0 & \sqrt{k}L^{-1} \end{pmatrix} \begin{pmatrix} b \\ a \end{pmatrix} \right\|_2 \rightarrow \min! \quad (5.3)$$

Jeśli obecność macierzy odwrotnej w powyższym sformułowaniu wciąż nas niepokoi, możemy dążyć dalej. Rozpisując, dostajemy inną postać minimalizowanego wyrażenia (5.3),

$$\|y - Xb - Za\|_2^2 + k\|L^{-1}a\|_2^2 \rightarrow \min!$$

Oznaczając  $g = L^{-1}a$ , mamy równoważnie

$$\|y - Xb - ZLg\|_2^2 + k\|g\|_2^2 \rightarrow \min!$$

— a tu już nie występuje macierz odwrotna do  $L$ ! Ostatecznie, dostajemy następujące sformułowanie zadania wyjściowego (por. [18, zadanie 11.3.4]:

1. Wyznacz rozkład Cholesky'ego  $A = LL^T$ .
2. Wyznacz rozwiązanie  $(b, g) \in \mathbb{R}^2 \times \mathbb{R}^n$  zadania najmniejszych kwadratów

$$\left\| \begin{pmatrix} y \\ 0 \end{pmatrix} - B \begin{pmatrix} b \\ g \end{pmatrix} \right\|_2 \rightarrow \min! \quad (5.4)$$

gdzie  $B$  jest prostokątną macierzą rozmiaru  $(m+n) \times (2+n)$  postaci

$$B = \begin{pmatrix} X & ZL \\ 0 & k^{1/2}I \end{pmatrix}.$$

3. Oblicz  $a = Lg$ .

## 5.2. Implementacja

Dzięki odpowiedniemu potraktowaniu problemu, całkowicie uniknęliśmy wyznaczania macierzy odwrotnych oraz niepotrzebnego przekształcania zadania najmniejszych kwadratów do postaci normalnej.

Rozkład Cholesky’ego macierzy  $A$  wyznaczymy korzystając z funkcji Octave  $U = \text{chol}(A)$ . Jednak musimy pamiętać, że wynikiem działania  $\text{chol}(A)$  jest macierz trójkątna *górną* taką, że  $U^T U = A$ , czyli innymi słowy,  $U = L^T$ . Dlatego musimy odpowiednio dostosować macierz zadania (5.4):

```
B = [X Z*U'; zeros(n,2), sqrt(k)*eye(size(U))];
```

Dalej, wystarczy rozwiązać zadanie najmniejszych kwadratów z macierzą  $B$ :

```
f = [y; zeros(n,1)];
q = B \ f;
b = q(1:2); a = U'*q(3:end);
```

Przypomnijmy, że w Octave zadanie najmniejszych kwadratów rozwiązuje się, korzystając z tego samego operatora „dzielenia”,  $\backslash$ , który służy do rozwiązywania układów równań.

Jednak nasze zadanie jest bardzo szczególnej postaci blokowej: blok  $(2, 2)$  macierzy  $B$  jest macierzą diagonalną, co może nam pozwolić osiągnąć dalszą redukcję kosztów obliczeń [18, rozdział 11.3]. Ponadto, można od razu tak ponumerować równania, by macierz  $Z$  była postaci

$$Z = \begin{pmatrix} 0 & I \end{pmatrix},$$

co spowoduje, że iloczyn  $ZL$  będzie wyznaczalny zerowym kosztem obliczeniowym.

## 5.3. Wykorzystanie specyfiki zadania

Dotychczas atakowaliśmy postawione zadanie przy minimalnej wiedzy o jego naturze. Staliśmy się przyjąć neutralny punkt widzenia numeryka, pozwalający nam dostrzec w zadaniu pewne typowe cechy samego zadania obliczeniowego. Jednak tym, co naprawdę jest piękne w obliczeniach naukowych jest to, że zadania — choć na swój sposób typowe — mają swoje niuanse, które powodują, że czasem warto zmienić swój punkt widzenia i dopasować używane metody do tego, co *więcej* wiemy o charakterze zadania!

Jak dotąd, braliśmy pod uwagę jedynie fakt, że macierz  $A$  jest z góry zadana, dosyć rzadka, symetryczna i dodatnio określona. Nie chcieliśmy wyznaczać  $A^{-1}$  wiedząc, jak niezręcznie jest numerycznie korzystać z takiej macierzy.

Przypomnijmy powody:

- Wyznaczenie macierzy odwrotnej  $A^{-1}$  jest procesem bardziej kosztownym niż wyznaczenie współczynników jej rozkładu (np. Cholesky’ego,  $A = LL^T$ , lub, unikając kosztownych pierwiastków,  $A = LDL^T$ )
- Aby dla danego  $y$  wyznaczyć  $A^{-1}y$  wystarczy rozwiązać dwa układy równań z macierzą  $L$ , każdy kosztem co najwyżej  $O(n^2)$  działań (i ewentualnie  $D$ , kosztem liniowym). Tak wyznaczone rozwiązanie numeryczne jest rozwiązaniem pewnego zadania sąsiedniego, o ile tylko użyliśmy dobrego algorytmu wyznaczania rozkładu macierzy.
- Nawet jeśli  $A$  jest macierzą rzadką, to  $A^{-1}$  zazwyczaj jest macierzą gęstą.

Tymczasem okazuje się, że spotykane w praktyce modele (5.1) korzystają z macierzy  $A$ , która jest tak zwaną macierzą addytywnych pokrewieństw. Mając zadaną taką macierz, możemy

wyznaczyć macierz do niej odwrotną, ale na pierwszy rzut oka nie widać w niej jakiegś wyrazistej regularności:

```
A = [0 0 0 1/2 0 1/2 1/4 1/4;
      0 0 0 0 1/2 1/2 1/4 1/4;
      0 0 0 0 1/2 0 1/4 1/2;
      0 0 0 0 0 1/4 1/2 1/8;
      0 0 0 0 0 1/4 1/2 3/8;
      0 0 0 0 0 0 1/4 1/2;
      0 0 0 0 0 0 0 1/4;
      0 0 0 0 0 0 0 0;];
n = size(A,1);
A = A + A' + eye(n);
inv(A)
```

Jednak, jeśli sprawdzić w fachowej literaturze (np. w [22]) *definicję* macierzy  $A$  to okaże się, że jej elementy wyznacza się z prostego rekurencyjnego wzoru, który bazuje na znanym z danych hodowlanych rodowodzie każdego zwierzęcia. Spojrzenie na wzór określający  $A$  w terminach operacji macierzowych, może słabszych psychicznie zwalić z nóg:

$$A = (I - P)^{-1}D(I - P)^{-T},$$

gdzie  $P$  jest pewną macierzą o co najwyżej dwóch niezerowych elementach w wierszu! Co więcej, macierz  $P$  bardzo łatwo wyznaczyć z danych rodowodowych, natomiast  $D$  jest zadaną macierzą *diagonalną* [22]. Stąd oczywiście dostajemy natychmiast *bardzo łatwo* wyliczalną macierz odwrotną,

$$A^{-1} = (I - P)^T D^{-1} (I - P).$$

Widzimy więc, że macierz  $A^{-1}$  nie dość, że jest banalna do wyznaczenia, to od razu jest dana w postaci rozkładu  $kA^{-1} = S^T S$ , w którym co prawda  $S = k^{1/2} D^{-1/2} (I - P)$  nie musi być trójkątna górna, ale za to na pewno jest bardzo rozrzedzona (ma tylko około  $2n$  niezerowych elementów).

To odkrycie zmienia nasz punkt widzenia! Jeśli bowiem mamy do dyspozycji rozrzedzoną macierz  $P$  i diagonalę  $d$  macierzy  $D$ , konstrukcja macierzy zadania najmniejszych kwadratów (5.2) upraszcza się do utworzenia bloku  $S$ , co możemy zaimplementować w Octave na przykład w poniższy sposób:

```
B = [X Z; zeros(n,2), spdiag(sqrt(k*d))*(speye(n)-P)];
```

#### 5.4. Przypadek dużego $n$

Niektóre badania mogą dotyczyć  $n = 10^6$  zwierząt ( $m$ , czyli zbiór danych pomiarowych) jest wtedy zwykle *dużo mniejsze*), więc każdy sposób na to, by obniżyć koszt pamięciowy i obliczeniowy wyznaczenia rozwiązania jest wart uwagi.

**Macierze rzadkie** Przede wszystkim, należy wykorzystać fakt, że macierze  $X, Z, A^{-1}$  są w praktyce macierzami mocno rozrzedzonymi.

Jeśli więc utworzymy  $X, Z, P$  jako macierze rzadkie (poleceniem **sparse**), to macierz  $B$  też będzie rzadka. W przypadku, gdy operator  $\backslash$  zostanie przyłożony do prostokątnej macierzy rzadkiej, spowoduje wywołanie specjalizowanej funkcji bibliotecznej z pakietu CXSPARSE, wykonującej rzadki rozkład QR.

**Zmniejszenie rozmiaru zadania przez powrót do układu równań normalnych** Równań normalnych nie musimy się obawiać, gdy uwarunkowanie macierzy  $B^T B$  jest nieduże. W niektórych przypadkach tak rzeczywiście będzie nawet dla bardzo dużych  $n$ . Należy jednak pamiętać, że jeśli  $m \ll n$ , to zastąpienie macierzy prostokątnej  $B$  rozmiaru  $(m+n) \times (2+n)$  macierzą kwadratową  $B^T B$  rozmiaru  $(2+n) \times (2+n)$  nie musi dawać znaczących zysków, zwłaszcza, że  $B^T B$  będzie mniej rozrzedzona niż  $B$ .

**Użycie metody iteracyjnej** Gdy  $n$  jest na tyle duże, że układ równań normalnych  $B^T B$  stanowiłby poważne wyzwanie dla metody bezpośredniej, można byłoby wówczas skorzystać z metody iteracyjnej rozwiązywania układu  $B^T B$  — na przykład, moglibyśmy tu zastosować metodę [PCG](#), jednak wyłącznie wówczas, gdy wyjściowe zadanie jest bardzo dobrze uwarunkowane.

Stosując metodę iteracyjną, można przy okazji uniknąć składania całej wielkiej macierzy  $A^{-1}$ : wystarczy, zgodnie z powyższym przepisem, przyłożyć macierz do wektora, co możemy zapisać stosunkowo prostą funkcją.

Alternatywą dla rozwiązywania układu równań normalnych metodą PCG może być rozwiązanie (nieco lepiej uwarunkowanego) zadania z (pozornie!) wielką macierzą kwadratową  $M$  rozmiaru  $2n + m + 2$ , o bardzo specjalnej, prostej strukturze:

$$M \begin{pmatrix} p \\ q \end{pmatrix} \equiv \begin{pmatrix} \alpha I & B \\ B^T & \end{pmatrix} \begin{pmatrix} p \\ q \end{pmatrix} = \begin{pmatrix} y \\ 0 \end{pmatrix},$$

gdzie  $\alpha > 0$  jest zadany parametrem dobranym do zadania.

Rzeczywiście,  $(p, q)$  jest rozwiązaniem powyższego równania wtedy i tylko wtedy, gdy  $q$  jest rozwiązaniem zadania najmniejszych kwadratów  $\|Bq - y\|_2 = \min$ , a  $p = (y - Bq)/\alpha$  jest przeskalowanym wektorem residuum.

Ponieważ macierz  $M$  jest symetryczna, ale nie jest dodatnio określona, należy zastosować do niej metodę PCR, dostępną m.in. w Octave. Prostoduszna implementacja

```
M = [alpha*speye(size(B,1)), B; B', spalloc(size(B,2),size(B,2))];
[X,info,relres] = pcr(M,[y;zeros(size(B,2),1)]);
info
q = X(size(B,1)+1:end);
```

może efektywnością ustąpić miejsca bardziej wyrafinowanej, korzystającej z operatorowej definicji  $M$ :

```
Mmult = @(x) [alpha*x(1:size(B,1)) + B*x(size(B,1)+1:end); B'*x(1:size(B,1))];
[X,info,relres] = pcr(Mmult,[y;zeros(size(B,2),1)]);
info
q = X(size(B,1)+1:end);
```

Koszt mnożenia wektora przez  $M$  możemy jeszcze bardziej zredukować, korzystając z wiedzy o strukturze  $B$ : jest ona macierzą blokową trójkątną górną, a i bloki mają specyficzną strukturę, upraszczającą mnożenie przez wektor.

**Obniżenie kosztu mnożenia przez  $A^{-1}$**  Możemy też próbować mnożenie przez  $A^{-1}$  uczynić tańszym, zwłaszcza, gdy implementujemy nasz program w języku C lub podobnym. Z pomocą przychodzi tu doświadczenie... programowania metody elementu skończonego (technika stosowana w numerycznych obliczeniach inżynierskich!). Ponieważ w wierszu  $P$  odpowiadającym  $i$ -temu zwierzęciu znajdują się co najwyżej dwie niezerowe wartości:

$$p_{i,s(i)} = p_{i,d(i)} = 1/2, \quad d_i = 1/2,$$

gdy ojcem  $i$  jest zwierzę o numerze  $s(i)$ , a matką — zwierzę o numerze  $d(i)$ , a w przypadku, gdy znane jest tylko jedno z rodziców,  $r(i)$ , tylko jeden element jest niezerowy,

$$p_{i,r(i)} = 1/2, \quad d_i = 3/4,$$

a gdy żadne z rodziców zwierzęcia  $i$  nie jest znane, cały  $i$ -ty wiersz  $P$  jest zerowy, natomiast  $d_i = 1$ .

Stąd wynika, że  $A^{-1} = \sum_i A_i^{-1}$ , gdzie  $A_i^{-1}$  jest wkładem  $i$ -tego zwierzęcia. Na przykład, gdy znani są oboje rodzice zwierzęcia,

$$A_i^{-1} = \begin{pmatrix} 1 \\ -1/2 \\ -1/2 \end{pmatrix} d_i^{-1} \begin{pmatrix} 1 & -1/2 & -1/2 \end{pmatrix},$$

przy czym elementy tej macierzy należy rozrzucić na trójkę  $i, s(i), d(i)$  współrzędnych  $A^{-1}$ .

**Ćwiczenie 5.1.** Niech będzie dana tabela  $R$ , rozmiaru  $n \times 2$ , określająca bezpośrednie relacje pomiędzy zwierzętami:

$$r_{i,1} = \begin{cases} s(i), & \text{gdy ojcem } i\text{-tego zwierzęcia był osobnik o numerze } s(i), \\ 0. & \end{cases}$$

Podobnie określamy  $r_{i,2}$ , identyfikator matki zwierzęcia  $i$ . Niech  $A$  rozmiaru  $n \times n$  będzie macierzą addytywnych pokrewieństw pomiędzy zwierzętami wyznaczoną przez  $R$ . Napisz możliwie szybko działającą procedurę, obliczającą  $A^{-1}y$  na zadanym wektorze  $y$  przy minimalnym zapotrzebowaniu na pamięć roboczą.

Jeśli chcesz zbliżyć się do *realnych* warunków pracy, przyjmij założenie, że dane z tablicy  $R$  są zapisane w arkuszu kalkulacyjnym Excela (lub w jakimś bardziej egzotycznym formacie): konwersja formatów danych to jeden z pomijanych, a bardzo uciążliwych programistycznie, aspektów obliczeń naukowych.

**Ćwiczenie 5.2.** W bardzo licznych zastosowaniach statystycznych należy wskazać zestaw tych wartości własnych (i, przy okazji, wektorów własnych) macierzy kowariancji  $X^T X$ , które są powyżej zadanego progu  $\eta^2$ . To zadanie nazywa się analizą głównych składowych (*principal component analysis*, PCA), a celem może być zmniejszenie rozmiaru zbioru danych statystycznych poprzez eliminację mało istotnych parametrów.

W sformułowaniu matematycznym, mając zadaną macierz  $X \in \mathbb{R}^{n \times m}$  — przy czym  $n \geq m$  — musimy wyznaczyć rozkład spektralny macierzy (symetrycznej i nieujemnie określonej)  $X^T X$ :

$$X^T X = V \Lambda V^T,$$

gdzie  $\Lambda$  jest macierzą diagonalną, zawierającą wartości własne, a  $V$  jest macierzą ortogonalną,  $V^T V = I$ , złożoną z wektorów własnych macierzy  $X^T X$ , a następnie zwrócić te wektory własne  $v_i$ , które spełniają warunek  $\lambda_i \geq \eta^2$ .

Napisz funkcję, która wykona to zadanie.

*Rozwiązanie.* Funkcja Octave, realizująca nasze zadanie mogłaby mieć postać:

```
function [V, L] = pca1(X, eta)
[V, L] = eig(X'*X);
[L, I] = sort(diag(L), 'descend'); V = V(:,I);
I = find(L>eta);
V = V(:,I); L = L(I);
end
```



Dodatkowo, nasza funkcja sortuje wektory i wartości własne w kolejności od największej, do najmniejszej.

Jednak ma ona tę wadę, że formując macierz  $X^T X$  tracimy informację zawartą w  $X$ : macierz  $X^T X$  jest wymiaru tylko  $m \times m$ . Dlatego bezpieczniej skorzystać z rozkładu SVD<sup>1</sup> macierzy  $X$ :

$$X = U \Sigma V^T$$

i faktu, że  $X^T X = V \Sigma^2 V^T$ . Ponieważ funkcja **svd** zwraca macierze  $U, \Sigma, V$ , lepsza numerycznie wersja naszej funkcji miałaby następującą postać:

```
function [V, L] = pca(X, eta)
[U L V] = svd(X,0); % economy-version
[L, I] = sort(diag(L).^2, 'descend'); V = V(:,I);
I = find(L>eta);
V = V(:,I); L = L(I);
end
```

---

<sup>1</sup> Więcej o rozkładzie według wartości szczególnych (SVD) dowiesz się z wykładu z [Matematyki Obliczeniowej II](#).

## 6. Charakterystyka pracy transformatora

Do transformatora o  $N$  zwojach doprowadzony jest prąd zmienny o napięciu  $E$  i częstotliwości  $\omega/2\pi$ . W modelowaniu jego pracy [8, Chapter 6, APP10] wykorzystuje się równanie różniczkowe opisujące zmiany natężenia prądu  $\phi$  w czasie:

$$\phi'' + \omega_0^2 \phi + b\phi^3 = \frac{\omega}{N} E \cos(\omega t). \quad (6.1)$$

Parametry  $\omega_0$  oraz  $b$  zależą od konkretnego transformatora. Podobne równanie rozwiązuje się, modelując np. zjawisko tzw. ferorezonansu w dużych sieciach transformatorów wysokiego napięcia, lecz wtedy czasem człon nieliniowy  $\phi^3$  zastępuje się wyższą nieparzystą potęgą  $\phi$ , np.  $\phi^{11}$  [21].

Niech warunek początkowy będzie postaci  $\phi(0) = \phi'(0) = 0$ . Naszym celem jest

- narysowanie wykresu  $\phi(t)$  dla  $t \in [0, 5]$  dla przypadku  $E = 165$ ,  $\omega = 120\pi$ ,  $N = 600$ ,  $\omega_0^2 = 83$ ,  $b = 0.14$ ;
- zbadanie wpływu niewielkich zmian parametru  $b$  na przebieg rozwiązania.

### 6.1. Bezstresowe rozwiązanie problemu

Aby rozwiązać (6.1), zastosujemy standardowy *solver* równań różniczkowych z Octave, *lsode*. Po sprowadzeniu równania do układu dwóch równań pierwszego rzędu,

$$\frac{d}{dt} \begin{pmatrix} \phi \\ v \end{pmatrix} = \begin{pmatrix} v \\ -\omega_0^2 \phi - b\phi^3 + \frac{\omega}{N} E \cos(\omega t) \end{pmatrix},$$

określamy funkcję wyznaczającą wektor prawej strony (przy umowie, że  $y = (y_1, y_2) = (\phi, v)$ ):

```
E = 165; omega = 120*pi; N = 600; omega02 = 83; b = 0.14;
oEN = (omega*E)/N;

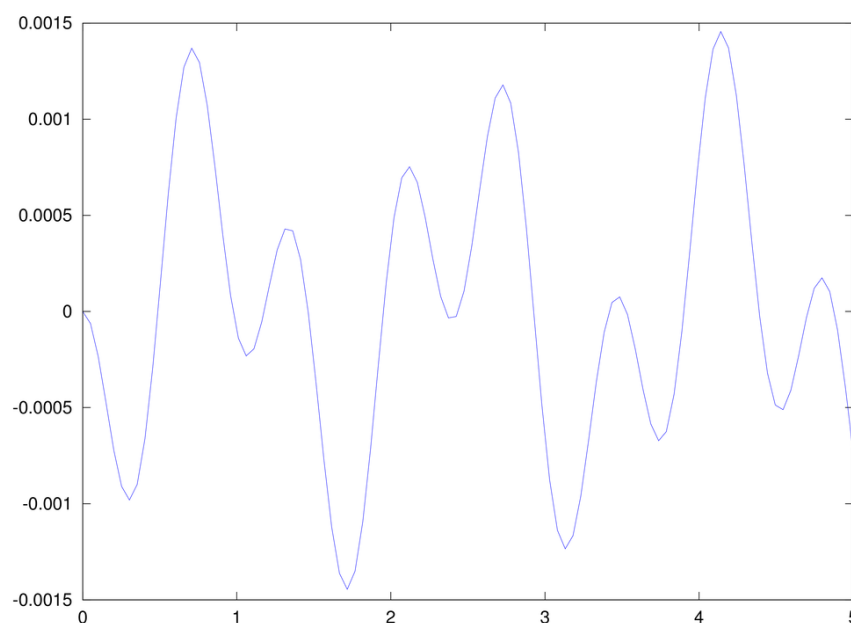
transf = @(y,t) [y(2); -y(1)*(omega02 + b*y(1)^2) + oEN*cos(omega*t)];
```

Numeryczne rozwiązanie i wykres  $\phi$  (czyli pierwszej współrzędnej wektora  $y$ ) wyznaczymy stosując proste komendy:

```
y0 = [0;0]; % warunek początkowy

K = 100; % rozdzielczość wykresu
t = linspace(0,5,K);

y = lsode(transf, y0, t);
plot(t, y(:,1)); % rysujemy tylko pierwszą współrzędną
```



Rysunek 6.1. Rozwiązanie równania transformatora? To się jeszcze okaże.

### 6.1.1. Próba weryfikacji rozwiązania

Uzyskany przez nas na rysunku 6.1 wykres — regularny, „optycznie prawie periodyczny” — zdaje się sensowny. Jednak — pamiętając o tym, że niektóre zadania mogą być numerycznie trudne, a ich rozwiązania zwodniczo eleganckie — na wszelki wypadek zastosujemy inżynierski chwyt, polegający na weryfikacji rozwiązania poprzez drastyczne zaostrożenie kryteriów tolerancji błędu.

Aby to osiągnąć, musimy nieco pomanipulować przy parametrach pracy `lsode`. Najpierw zapamiętamy bieżące wartości tolerancji błędu<sup>1</sup> względnego i bezwzględnego:

```
rtol = lsode_options('relative_tolerance');
atol = lsode_options('absolute_tolerance');
```

Następnie ustalimy nowe wartości na poziomie  $10^4$  razy(!) mniejszym:

```
lsode_options('relative_tolerance', rtol*1e-4);
lsode_options('absolute_tolerance', atol*1e-4);
```

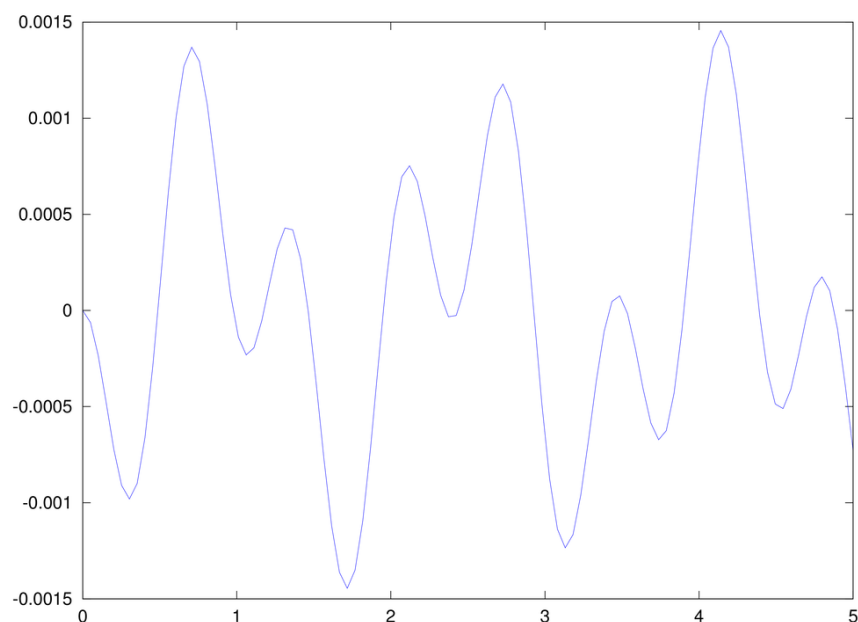
i na nowo uruchomimy tą samą symulację:

```
y = lsode(transf, y0, t);
plot(t, y(:,1));

% na zakończenie przywracamy poprzednie wartości parametrów lsode
lsode_options('relative_tolerance', rtol);
lsode_options('absolute_tolerance', atol);
```

<sup>1</sup> Przypomnijmy na wszelki wypadek, że ustalenie wartości tych parametrów nie spowoduje bynajmniej, że uzyskane rozwiązanie będzie obciążone właśnie takim błędem; faktyczna rola parametrów tolerancji błędu jest opisana specjalistycznej literaturze, np. w [10].

Choć na wyniki przyjdzie nam czekać znacznie dłużej niż poprzednio — wszak *utrudniliśmy pracę solverowi!* — efekt, który możemy sprawdzić na rysunku 6.2, dość przekonująco potwierdza, że *solver* nie oszukał nas poprzednim razem i wyznaczone rozwiązanie jest praktycznie identyczne.



Rysunek 6.2. Numeryczne potwierdzenie jakości rozwiązania równania transformatora? „Na oko” zdaje się, że jest OK.

Zatem na podstawie uzyskanego wykresu możemy skonkludować, że nasz transformator ma na tyle dużą bezwładność, że — choć w jego pracy pojawiają się oscylacje — przebiegają one łagodnie. W badanym okresie czasu natężenie tylko dwa razy (pięć, jeśli darować sobie drobne różnice) przyjęło maksymalną wartość.

## 6.2. Czy interpretacja wyniku jest poprawna?

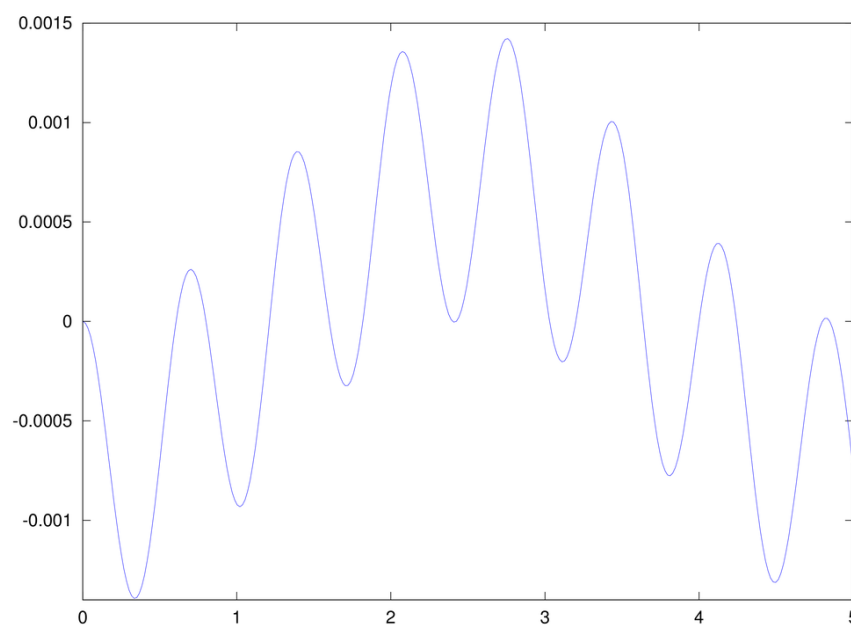
Jednak otrzymany przez nas wykres jest — w potocznym rozumieniu — *całkowicie błędny!* Zapomnieliśmy bowiem o innym zjawisku: o *aliasingu*, czyli niebezpieczeństwie wyciągnięcia błędnych wniosków o przebiegu funkcji w przypadku zbyt małej liczby punktów próbkowania jej wartości. Rzeczywiście, powinni byliśmy na wszelki wypadek zbadać, czy zwiększenie *rozdzielczości* wykresu nie spowoduje jego wyraźnej zmiany....

Zatem zobaczmy. Wybierzemy — zamiast  $K = 100$  punktów wykresu, jak dotychczas — na początek  $K = 300$  równoodległych punktów:

```
K = 300;
t = linspace(0,5,K);

y = lsode(transf, y0, t);
plot(t, y(:,1));
```

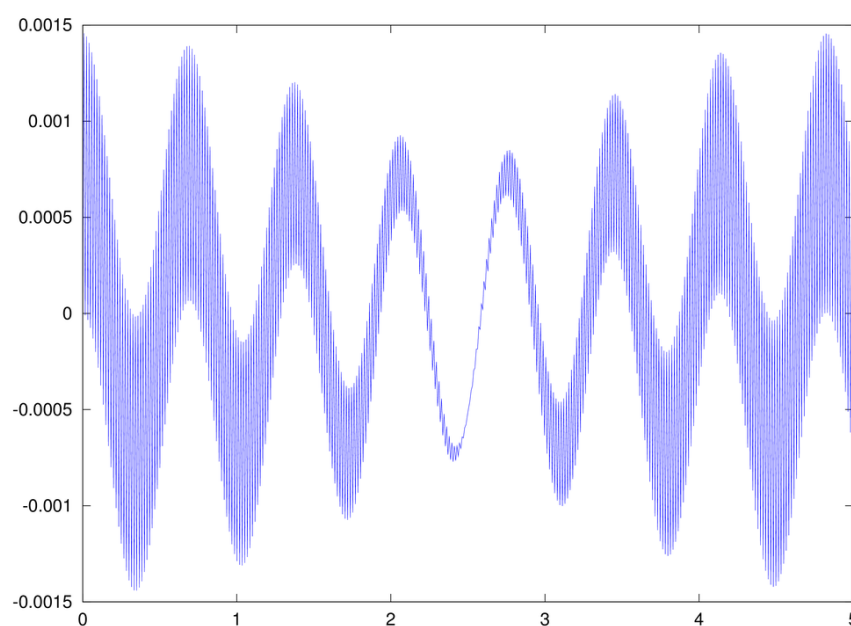
Hmmm... dostaliśmy wykres (zob. rysunek 6.3), w żaden sposób nie przypominający rysunku 6.1, który jeszcze przed chwilą wydawał nam się tak sensowny... A przecież jedynym



Rysunek 6.3. Rozwiązanie równania transformatora próbkowane w  $K = 300$  punktach.

zmienionym parametrem była liczba punktów próbkowania rozwiązania: zatem diagnoza jest jasna: faktycznie, mamy *aliasing*.

No dobrze, zatem który z wykresów jest prawidłowy? Możemy powtórzyć nasz chwyt i próbkować w  $K = 600$  punktach (zob. rysunek 6.4), ale... może lepiej postąpić nieco bardziej metodycznie, zwłaszcza, że za chwilę możemy natknąć się na barierę rozdzielczości... naszego ekranu komputerowego?



Rysunek 6.4. Rozwiązanie równania transformatora próbkowane w  $K = 600$  punktach.

Już przekonaliśmy się, że (zapewne) nasz *solver*, *lsode*, nie ma większych kłopotów z aprok-

symacją rozwiązania. Pozostaje więc przekonać się, kiedy niewielkie zwiększanie (lub, co — w świetle powyższego — na jedno wychodzi, *zmniejszenie*) rozdzielczości wykresu przestanie powodować istotną zmianę *wykresu* rozwiązania.

### 6.2.1. Jak dobrać dobrą rozdzielczość?

Gdybyśmy bowiem **wcześniej** choć **chwilę zastanowili się** nad charakterem naszego równania, zobaczylibyśmy, że jego prawa strona — zewnętrzne wymuszenie — jest postaci  $\cos(\omega t)$ , czyli, dla naszych danych, postaci  $\cos(120 \pi t)$ . Nawet intuicyjnie jest jasne, że próbkowanie wartości rozwiązania powinniśmy dopasować do tej częstości. W interesującym nas przedziale czasowym  $t \in [0, 5]$  takie wymuszenie będzie miało  $120 \cdot 5 = 700$  ekstremów, więc — biorąc skromnie po 10 węzłów pomiędzy jednym a drugim ekstremum — powinniśmy użyć około 7000 węzłów, by ustawić wstępną (minimalną sensowną?) rozdzielczość wizualizacji.

Naturalnie, możemy dodatkowo oprzeć się na naszej wiedzy dla liniowych równań różniczkowych. W przypadku  $b = 0$  drgania będą miały dwie składowe: wolnozmienną postaci  $\cos(\omega_0 t)$  (być może z jakimś przesunięciem fazowym), związaną z rozwiązaniem ogólnym jednorodnego równania oscylatora harmonicznego — oraz składową szybkozmienną, postaci  $\cos(\omega t)$ , będącą rozwiązaniem szczególnym równania niejednorodnego. W naszym przypadku, da to coś w stylu  $A_0 \cos(9.11 t + \phi) + A_1 \cos(376.99 t)$  — zatem faktycznie musimy dostosować rozdzielczość wizualizacji co najmniej do najszybciej zmieniającego się składnika,  $\cos(\omega t)$ .

Aby utwierdzić się w naszych przypuszczeniach, zrealizujemy w pętli wyświetlanie rozwiązania, oryginalnie wypróbkowanego w bardzo wielu punktach ( $K = 5000$ ), nie zważając na rozdzielczość monitora, która zapewne nie przekracza 2000 pikseli w poziomie, na coraz rzadszej siatce węzłów:

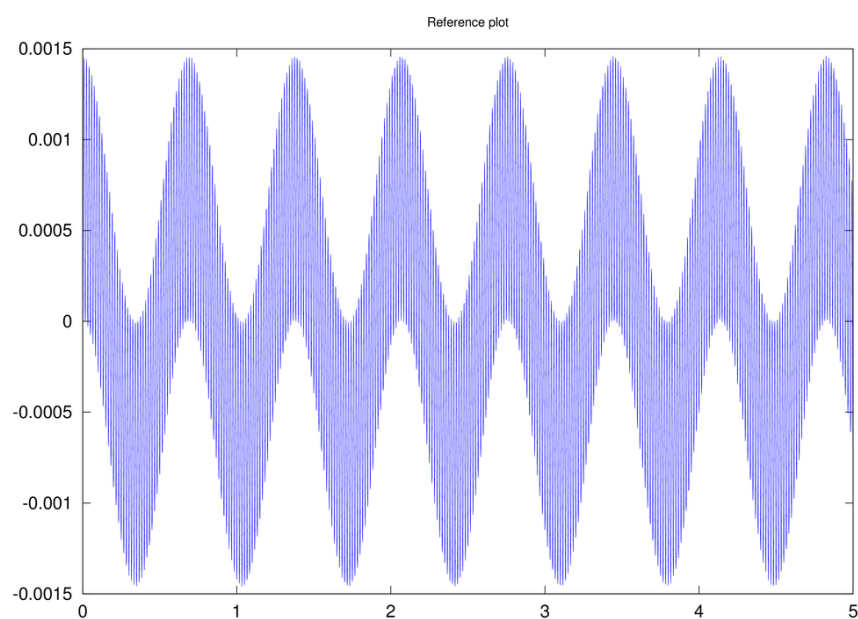
```
K = 5000;
t = linspace(0,T,K); % highest possible resolution
y = lsode(transf, y0, t); % reference solution
plot(t,y(:,1)); title('Reference_plot'); pause(3);

P = 1;
for k = [1:10]
    plot(t(P:k:end),y(P:k:end,1));
    title(['Reference_plot_for_K=',num2str(K),'_every_',num2str(k),'th_point']); pause(3);
end
```

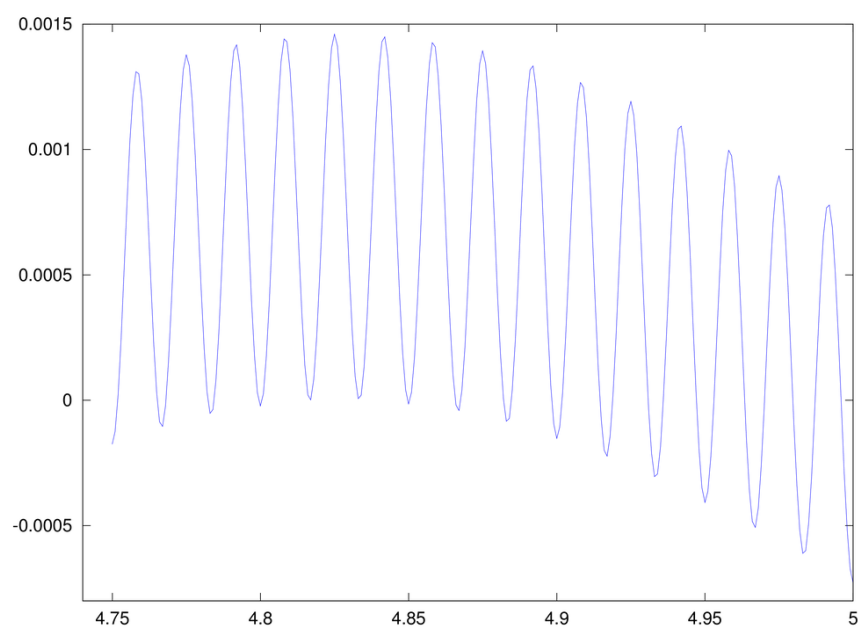
Ta seria wykresów przekonuje nas jasno, że rozwiązanie zmienia się *zbyt szybko*, by było czytelnie wizualizowane na odcinku  $[0, 5]$  — zob. rysunek 6.5. Ponadto faktycznie wydaje się, że obserwujemy zjawisko nakładania się szybkozmiennnej składowej na składową wolnozmienną. Obie te obserwacje skłaniają więc nas do tego, by przyjrzeć się dokładniej wykresowi ograniczonemu do znacznie węższego zakresu  $t$ : wybieramy „na oko” okno o szerokości 5% oryginalnego przedziału, zlokalizowane na jego końcu.

```
P = floor(0.95*K); % ostatnie 5% wykresu
for k = [1:10]
    plot(t(P:k:end),y(P:k:end,1));
    title(['Reference_plot_for_K=',num2str(K),'_every_',num2str(k),'th_point']); pause(3);
end
```

Strzał okazał się znakomity (rysunek 6.6) — wydaje się, że przy oryginalnej rozdzielczości  $K = 5000$  (której w zawężonym przedziale długości 0.25 odpowiadało 250 węzłów) udało się zlokalizować najważniejsze cechy rozwiązania: wszystko bowiem wskazuje na to, że rozwiązanie ma w badanym zawężonym przedziale około 20 równomiernie rozmieszczonych ekstremów.



Rysunek 6.5. Rozwiązanie równania transformatora próbkowane w  $K = 5000$  punktach.



Rysunek 6.6. Rozwiązanie równania transformatora próbkowane w  $K = 5000$  punktach, zobrazone na krótkim przedziale.

Możemy teraz potwierdzić swoje przypuszczenie, biorąc  $K = 20000$  (czemu odpowiada 1000 węzłów w zawężonym przedziale) i sprawdzając tam przebieg rozwiązania.

**Ćwiczenie 6.1.** Jak oglądać ostatnie 5% rozwiązania, ale bez  $K$  idącego w tysiące (dziesiątki tysięcy)?

```
E = 165; omega = 120*pi; N = 600; omega02 = 83; b = 0.14;
oEN = (omega*E)/N;
```

```

transf = @(y,t) [y(2); -y(1)*(omega02 + b*y(1)^2) + oEN*cos(omega*t)];
y0 = [0;0]; T = 5;

% high voltage transformer
% E = 0.15*63.5e3; C = 777; R = 48.4e3;
% omegaS = 377/(2*pi); omega02 = (53/(2*pi))^2; omega22 = (85/(2*pi))^2;

```



To tylko fragment kodu źródłowego Octave. Więcej na <http://mst.mimuw.edu.pl/lecture.php?lecture=ona&part=Ch6>.

### 6.3. Eksperymenty z parametrami układu

Naszym drugim celem jest zbadanie wrażliwości rozwiązania na małe zmiany  $b$ . Otóż korzystając z poprzednio opracowanego kodu i stopniowo zmniejszając  $b$  aż do 0 widzimy, że rozwiązanie różni się niewiele od dotychczas rozważanego. Możemy więc *przypuszczać*, że (dla małych wartości  $b$ ) rozwiązania zależą w sposób ciągły od tego parametru, a linearyzacja modelu jest tu *najprawdopodobniej* uzasadniona.

Tu kończy się etap obliczeń naukowych — uprawiania „matematyki” eksperymentalnej: teraz powinniśmy przejść do dowodzenia twierdzeń. Czy jesteś w stanie *udowodnić* powyższe własności rozważanego równania?

### 6.4. Uwagi i uzupełnienia

Warto na zakończenie wspomnieć, że — choć sprawiły nam niejakie kłopoty — rozwiązania naszego równania przy danych wartościach parametrów zachowywały się bardzo przyzwoicie. Tymczasem dla innych zestawów parametrów może być znacznie gorzej. Równanie tej postaci jest znane pod nazwą równania Duffinga (z periodycznym wymuszeniem). Modeluje ono nie tylko pracę transformatora, ale także działanie pewnych układów mechanicznych. W ogólnym przypadku jest ono postaci

$$\phi'' + \epsilon\phi' \pm \omega_0^2\phi + \delta\phi^3 = f(t)$$

i jest znany przykład zadania różniczkowego, które dla pewnych wartości  $\epsilon$  i  $\omega_0$  może mieć **chaotycznie** zachowujące się rozwiązania, a to oznacza, że nieunikniony błąd numerycznej aproksymacji szybko spowoduje przeskok na odległą od rzeczywistej trajektorii i całkowitą utratę jakości rozwiązania.

**Ćwiczenie 6.2.** W modelu sieci energetycznej z transformatorami wysokonapięciowymi pojawia się równanie opisujące wartość natężenia prądu  $\phi(t)$  w chwili  $t$ , postaci [21]

$$\phi'' + \frac{1}{RC}\phi' + \omega_0^2\phi + \omega_2^2\phi^3 = \omega E \cos(\omega t).$$

Równanie uzupełnione jest warunkiem początkowym  $\phi(0) = \dots, \phi'(0) = 0$ . Dla wartości parametrów:  $R = 48.4$  [kΩ],  $C = 777$  [nF],  $E = 0.15 \cdot 63.5$  [kV],  $\omega = 377$  [rad/s],  $\omega_0 = 53$  [rad/s],  $\omega_2 = 85$  [rad/s], wyznacz przebieg  $\phi$  w ciągu pierwszych 5 minut.



## 7. Linie i okręgi

Podczas kursu matematyki obliczeniowej zetknęliśmy się z zadaniem: mając dany zestaw punktów na płaszczyźnie:

$$P = \{p_i = (x_i, y_i) \in \mathbb{R}^2 : i = 0, \dots, N\},$$

znaleźć prostą  $L(x) = ax + b$  taką, że

$$\sum_{i=0}^N |L(x_i) - y_i|^2 = \min!$$

Było to po prostu inne sformułowanie liniowego zadania najmniejszych kwadratów na współczynniki  $a, b$ . Zwróćmy jednak uwagę, że takie sformułowanie zadania nierówno traktuje zmienne  $x_i$  oraz  $y_i$ .

W wielu zastosowaniach, na przykład: w analizie obrazów, w energetyce, czy w komunikacji, mamy do czynienia z zadaniem minimalizacji średniej odległości prostej od zestawu  $P$  punktów na płaszczyźnie. Rzeczywiście, jest to matematyczne sformułowanie praktycznego zadania w rodzaju: *Jak poprowadzić (prostą) autostradę tak, by dojazd do niej z pobliskich miejscowości był możliwie szybki?* Albo: *Jak, na podstawie nieco rozmytego konturu budynku na zdjęciu, zdefiniować jego brzeg?*

Przez analogię do zadania najmniejszych kwadratów określimy zadanie *geometrycznego dopasowania* prostej do punktów, polegające na wskazaniu takiej prostej  $L$ , dla której

$$\sum_{i=1}^N \text{dist}(p_i, L)^2 = \min!$$

przy czym odległość będziemy mierzyć w normie euklidesowej.

Powyższe zadanie jest najprostszą wersją całej klasy zadań geometrycznego dopasowania, w których krzywą  $\gamma$  określonego typu chcemy dopasować do zadanego zestawu punktów  $P$  tak, by

$$\sum_{i=1}^N \text{dist}(p_i, \gamma)^2 = \min!$$

W przypadku budynków na przykład, szukalibyśmy prostokąta spełniającego powyższy warunek (por. uwagę na końcu wykładu). W niniejszym rozdziale zajmiemy się na początek, w oparciu o [7], geometrycznym dopasowaniem prostej, a w dalszej części wykładu, korzystając z materiałów zawartych w [1] — dopasowaniem okręgu do zadanych punktów.

Liczba punktów pomiarowych  $N$  w praktyce nie jest bardzo duża, dlatego będziemy mieli do czynienia z zadaniami niezbyt intensywnymi obliczeniowo: głównym problemem będzie tutaj właściwe sformułowanie zadania w taki sposób, by można było je skutecznie zaatakować dostępnymi metodami.

### 7.1. Dopasowanie prostej

Nietrudno zauważyć, że reprezentowanie prostej  $L$  w postaci  $L(x) = ax + b$  nie jest zbyt zgrzeszne, bo na przykład czasem najlepsze dopasowanie może zachodzić dla prostej  $x = 0$ , której

nie jesteśmy w stanie tak przedstawić. Dlatego korzystniej będzie rozpatrzeć ogólne równanie prostej  $L$  postaci

$$Ax + By + C = 0,$$

oczywiście uzupełnione jakimś warunkiem normującym (współczynniki  $A, B, C$  są dane z dokładnością do mnożnika). Tutaj najrozsądniej będzie przyjąć, że  $A^2 + B^2 = 1$ : wektor  $[A, B]$  jest wektorem normalnym do prostej  $L$ . Wtedy odległość punktu  $p_i = (x_i, y_i)$  od  $L$  jest dana równością

$$\text{dist}(p_i, L) = \frac{|Ax_i + By_i + C|}{\sqrt{A^2 + B^2}} = |Ax_i + By_i + C|,$$

zatem mamy znaleźć  $A, B, C$  takie, że  $A^2 + B^2 = 1$  oraz wyrażenie

$$f(A, B, C) = \sum_{i=1}^N (Ax_i + By_i + C)^2$$

przyjmuje najmniejszą wartość.

### 7.1.1. Uproszczenia i analiza problemu

Ponieważ na parametr  $C$  nie ma ograniczeń, możemy łatwo go wyeliminować z rozważań przyrównując do zera pochodną

$$\frac{\partial f}{\partial C} = 2 \sum_{i=1}^N (Ax_i + By_i + C) = 0 \quad \implies \quad C = -(A\bar{x} + B\bar{y}),$$

gdzie  $\bar{x} = \frac{1}{N} \sum_{i=1}^N x_i$  i analogicznie  $\bar{y} = \frac{1}{N} \sum_{i=1}^N y_i$ .

W ten sposób zadanie zredukowało się do znalezienia wektora  $u = (A, B)^T$  takiego, że

$$\|Mu\|_2^2 = \min!, \quad \text{przy czym } \|u\|_2^2 = 1,$$

gdzie

$$M = \begin{pmatrix} x_1 - \bar{x} & y_1 - \bar{y} \\ x_2 - \bar{x} & y_2 - \bar{y} \\ \vdots & \vdots \\ x_N - \bar{x} & y_N - \bar{y} \end{pmatrix}.$$

Jest to więc nic innego, jak zadanie wyznaczenia wektora szczególnego<sup>1</sup> macierzy  $M$ , odpowiadającego najmniejszej wartości szczególnej. Rzeczywiście, jeśli  $M = U\Sigma V^T$  i macierze  $U, V$  są ortogonalne, a

$$\Sigma = \begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \end{pmatrix},$$

przy czym dla ustalenia uwagi  $\sigma_1 \geq \sigma_2 \geq 0$ , to

$$\|Mx\|_2^2 = \|U\Sigma V^T x\|_2^2 = \|\Sigma V^T x\|_2^2 = x^T V \begin{pmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{pmatrix} V^T x \geq \sigma_2^2 \|V^T x\|_2^2 = \sigma_2^2 \|x\|_2^2 = \sigma_2^2,$$

a równość zachodzi gdy  $V^T x = [0, 1]^T$ , czyli dla  $x = v_2$ .

<sup>1</sup> Więcej na temat wektorów i wartości szczególnych można dowiedzieć się np. w wykładzie z [Matematyki Obliczeniowej II](#), w rozdziale dotyczącym rozkładu SVD macierzy.

### 7.1.2. Implementacja

Ponieważ w Octave znajduje się gotowa funkcja wyznaczająca pełny rozkład SVD zadanej macierzy, możemy pokusić się o implementację funkcji wyznaczenia  $A, B, C$  na podstawie punktów, których współrzędne na osiach  $OX$  i  $OY$  podane są, odpowiednio, w talicach  $x$  i  $y$ :

```
function ABC = linefit(x,y)
x = x(:); y = y(:);
ABC = NaN(3,1);
xm = mean(x); ym = mean(y);
[U, S, V] = svd([x-xm, y-ym], 0); % ekonomiczny rozkład SVD
ABC(1:2) = V(:,end); % [A, B]
ABC(3) = - ABC(1)*xm - ABC(2)*ym;
end
```

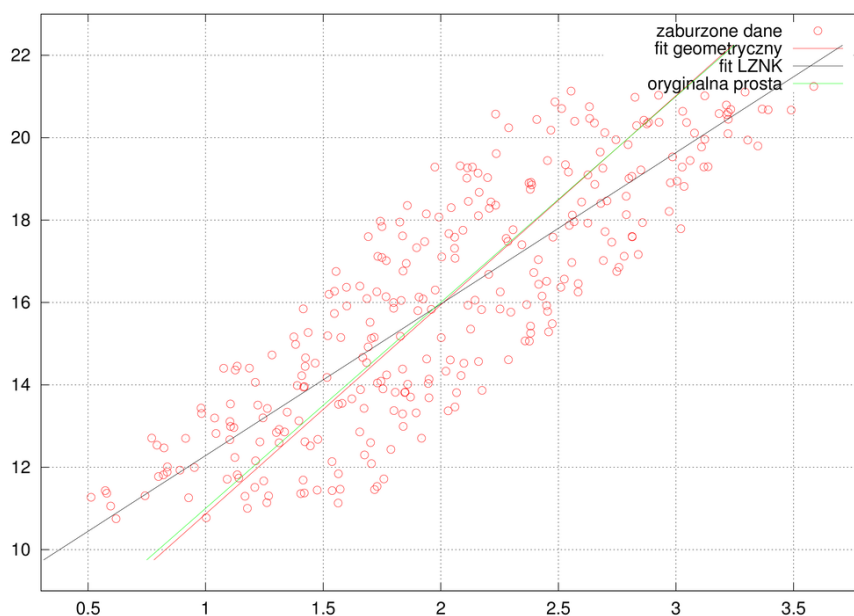
Aby przetestować działanie naszego kodu, zaburzymy losowo punkty prostej  $y = 5x + 6$  i sprawdzimy, co wyjdzie. Porównamy także nasz (geometryczny) *fit* z dopasowaniem na podstawie standardowej metody najmniejszych kwadratów.

W celu wygodnego rysowania prostej zadanej w postaci uwikłanej, naprędce opracujemy funkcję lineplot, której listing zamieszczamy poniżej.

```
function lineplot(ABC, X, Y, color)
if nargin < 4
    color = '';
end
xmin = X(1); xmax = X(2);
ymin = Y(1); ymax = Y(2);
A = ABC(1); B = ABC(2); C = ABC(3);
if abs(A/B) <= 1
    plot(X, (-C-A*X)/B, ['- ',color]);
else
    plot((-C-B*Y)/A, Y, ['- ',color]);
end
end
```

Teraz możemy rozpocząć testy, korzystając na przykład z kodu poniższej postaci:

```
a = 5; b = 6;
x = linspace(1,3,300);
y = a*x + b;
x = x+0.6*(2*rand(size(x))-1);
y = y+0.6*(2*rand(size(y))-1);
% fit geometryczny
ABC = linefit(x,y);
% fit LZNK
ba = [ones(size(x)); x]'\y';
plot(x,y,'or');
hold on;
lineplot(ABC,[0,4],[min(y)-1,max(y)+1],'r');
lineplot([ba(2),-1,ba(1)],[0,4],[min(y)-1,max(y)+1],'k');
lineplot([a,-1,b],[0,4],[min(y)-1,max(y)+1],'g');
hold off;
legend('zaburzone_dane','fit_geometryczny','fit_LZNK','oryginalna_prosta');
grid on;
```



Rysunek 7.1. Przykładowy wynik dopasowania prostej w sensie LZNK i w sensie geometrycznym.

## 7.2. Dopasowanie okręgu

Analogicznie, można rozważyć zadanie geometrycznego dopasowania okręgu (lub, w większej ogólności, elipsy)  $S$  do danych punktów na płaszczyźnie. Zastosowania mogą dotyczyć [1] medycyny (zwłaszcza — okulistyki), kontroli jakości wykonania obrotowych części mechanicznych, wyznaczania torów cząstek elementarnych, robotyki, a nawet — archeologii (wyznaczenie średnicy stadionu na podstawie zachowanych fragmentów). W wymienionych przypadkach zazwyczaj mamy do czynienia z zestawem punktów znajdujących się na łuku stanowiącym jedynie niewielki fragment okręgu. Z powodu zaburzeń danych, spowodowanych na przykład błędami pomiaru, zadane punkty nie układają się idealnie na poszukiwanej krzywej.

Jeśli okrąg  $S$  reprezentować przez środek  $O = (a, b)$  oraz promień  $r$ :

$$S = \{(x, y) \in \mathbb{R}^2 : (x - a)^2 + (y - b)^2 = r^2\},$$

to odległość  $d_i$  danego punktu  $p_i = (x_i, y_i)$  od  $S$  jest równa, jak łatwo sprawdzić,

$$d_i = |||p_i - O||_2 - r|.$$

W takim razie, musimy zminimalizować wartość funkcjonału

$$f(a, b, r) = \sum_{i=1}^N \left( \sqrt{(x_i - a)^2 + (y_i - b)^2} - r \right)^2.$$

### 7.2.1. Atak na wprost

Ponieważ zadanie minimalizacji tym razem dotyczy bardziej skomplikowanego funkcjonału nieliniowego, użyjemy standardowej procedury optymalizacji nieliniowej w Octave, `sqp`.

```
function d = residgeom(X,x,y)
    % a = X(1); b = X(2); r = X(3);
```

```

d = sumsq( sqrt((x-X(1)).^2 + (y-X(2)).^2) - X(3));
end

function [S,dist,info] = fitcircle1(S0,x,y)
% S = (a,b,r)
[S,dist,info] = sqp(S0,@(X)residgeom(X,x,y));
end

```

Funkcja `residgeom` wyznacza wartość funkcjonału  $f(a, b, r)$ , przy czym przekazujemy jej wszystkie parametry funkcjonału w pierwszym argumencie, w postaci wektora  $X$ . Musimy tak zrobić, bo `sqp` spodziewa się funkcji *jednego* argumentu — taką tworzymy w postaci funkcji anonimowej

```
@(X)residgeom(X,x,y)
```

przekazywanej do `sqp`. Ponieważ w zadaniu nieliniowym wymagany jest sensowny punkt startowy  $S_0$  — taki więc musimy przekazać, prócz zadanych współrzędnych punktów  $(x_i, y_i)$ , do funkcji `fitcircle1`. Zadaniem użytkownika jest *właściwy* dobór  $S_0 = (a_0, b_0, r_0)$  tak, by metoda iteracyjna była zbieżna do *właściwego* minimum.

### 7.2.2. Zmiana sformułowania zadania

Jakkolwiek dokonany przez nas wybór parametrów zadania:  $a, b, r$ , jest niewątpliwie kuszący, to jednak, gdy punkty  $(x_i, y_i)$  są prawie współliniowe, wyznaczany promień  $r$  może być bardzo duży w porównaniu do współrzędnych środka  $(a, b)$ . Ponadto małe zaburzenia położenia punktów  $(x_i, y_i)$  mogą prowadzić do dużych skoków wielkości  $r$  (a także — położenia środka okręgu), dlatego warto zadanie sformułować inaczej.

Okrąg  $S$  możemy opisać za pomocą standardowego równania krzywej drugiego stopnia,

$$A(x^2 + y^2) + Bx + Cy + D = 0.$$

Ponieważ wynika stąd, że

$$\left(x + \frac{B}{2A}\right)^2 + \left(y + \frac{C}{2A}\right)^2 = \frac{B^2 + C^2 - 4AD}{4A^2},$$

Jak podaje [1], Pratt zasugerował warunek normalizacyjny

$$B^2 + C^2 - 4AD = 1.$$

Ponadto wówczas odległość  $d_i$  punktu  $(x_i, y_i)$  od okręgu jest równa

$$d_i = 2 \frac{r_i}{1 + \sqrt{1 + 4Ar_i}}, \text{ gdzie } r_i = A(x_i^2 + y_i^2) + Bx_i + Cy_i + D,$$

zatem musimy minimalizować funkcjonal

$$f(A, B, C, D) = \sum_i d_i^2$$

z ograniczeniem

$$g(A, B, C, D) \equiv B^2 + C^2 - 4AD - 1 = 0.$$

To zadanie po raz kolejny zrealizujemy za pomocą funkcji `sqp`:

```

function d = residgeom2(X,x,y)
    A = X(1); B = X(2); C = X(3); D = X(4);
    P = A*(x.^2 + y.^2) + B*x + C*y + D;
    d = sumsq(2*(P./(1+sqrt(1+4*A*P))));
end

function d = constr(X)
d = X(2)^2 + X(3)^2 - 4*X(1)*X(4) - 1;
end

function [S,dist,info] = fitcircle2(S0,x,y)
% S = (A,B,C,D)
c = zeros(4,1);
c(1) = 1/(2*S0(3)); c(2:3) = -2*c(1)*S0(1:2); c(4) = (c(2)^2 + c(3)^2 - 1)/(4*c(1));

[c,dist,info] = sqp(c, @(X)residgeom2(X,x(:),y(:)), @constr);
S = [-c(2:3)/(2*c(1)); sqrt(c(2)^2 + c(3)^2 - 4*c(1)*c(4))/(2*abs(c(1)))];
end

```

Jakkolwiek ostatnią liniijkę kodu moglibyśmy teoretycznie zastąpić tańszym  $S = [-c(2:3); 1]/(2*c(1))$ , to jednak bezpieczniej będzie zrobić tak, jak powyżej — bo warunek  $B^2 + C^2 - 4AD = 1$  może nie być idealnie spełniony.

### 7.2.3. Wybór przybliżenia początkowego jako nowe zadanie

Pozostaje jeszcze pytanie, w jaki sposób będziemy mogli sensownie wyznaczyć początkowe przybliżenie  $S_0$ . Nadspodziewanie dobrym kandydatem na  $S_0$  jest (patrz [1]) okrąg spełniający tzw. *algebraiczny warunek dopasowania*, tzn. minimalizujący, przy ograniczeniu  $B^2 + C^2 - 4AD = 1$ , sumę kwadratów residuum równania okręgu:

$$\mathcal{F}(A, B, C, D) = \sum_i r_i^2,$$

gdzie  $r_i$  jest zadane przez  $r_i = A(x_i^2 + y_i^2) + Bx_i + Cy_i + D$ .

Jest to więc — bardzo podobnie jak widzieliśmy to w przypadku fitowania prostej — zadanie najmniejszych kwadratów dla  $u = (A, B, C, D)$

$$\|Mu\|_2^2 = \min!, \text{ z ograniczeniem } u^T Ku = 1,$$

gdzie

$$M = \begin{pmatrix} x_1^2 + y_1^2 & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ x_n^2 + y_n^2 & x_n & y_n & 1 \end{pmatrix}, \quad K = \begin{pmatrix} & & & -2 \\ & 1 & & \\ & & 1 & \\ -2 & & & \end{pmatrix}_{4 \times 4}.$$

Niestety, macierz  $K$  nie jest dodatnio określona, co komplikuje algorytm rozwiązywania tego zadania.

Funkcja Lagrange'a jest postaci

$$L(u, \lambda) = u^T M^T M u - \lambda(u^T K u - 1),$$

zatem koniecznym warunkiem dla ekstremum jest

$$L_u(u, \lambda) = 0 \text{ oraz } u^T K u = 1.$$

Ponieważ  $L_u(u, \lambda) = M^T M u - \lambda K u$ , to pierwszy warunek oznacza po prostu, że para  $(\lambda, u)$  jest parą własną powyższego uogólnionego zadania własnego,

$$M^T M u - \lambda K u = 0. \quad (7.1)$$

Wprowadzając (ekonomiczny) rozkład QR macierzy  $M$ ,

$$M = QR,$$

mamy równoważnie  $R^T R u - \lambda K u$  i stąd  $(Ru, \lambda)$  musi być parą własną macierzy  $RK^{-1}R^T$ . Ponieważ macierz  $K^{-1}$  ma wartości własne  $\{1, 1, 1/2, -1/2\}$ , to macierz  $RK^{-1}R^T$  (na mocy tw. Sylwestera) także ma dokładnie jedną wartość własną ujemną, a pozostałe trzy — dodatnie.

Mnożąc stronami (7.1) przez  $u^T$  dostajemy

$$\lambda = \frac{u^T M^T M u}{u^T K u} = \frac{\|Mu\|_2^2}{u^T K u} = \|Mu\|_2^2,$$

na mocy warunku  $u^T K u = 1$ . Ponieważ  $\|Mu\|_2^2 \geq 0$ , interesująca nas  $\lambda$  nie może być ujemna. Ze względu na to, że poszukiwane przez nas  $u$  musi minimalizować  $\|Mu\|_2^2$ , znaczy to, że minimum zostanie przyjęte równe najmniejszej *dodatniej* wartości własnej  $B$ . W takim razie szukana przez nas para własna to para odpowiadająca najmniejszej dodatniej wartości własnej macierzy  $RB^{-1}R^T$ .

Stąd jednym ze sposobów implementacji powyższego algorytmu wyznaczenia  $u$  — a tym samym okręgu spełniającego wymieniony na początku tej sekcji algebraiczny warunek dopasowania — może być funkcja:

```
function [S,dist,info] = fitcircle3(x,y)
x = x(:); y = y(:);
invB = [0 0 0 -0.5; 0 1 0 0; 0 0 1 0; -0.5 0 0 0]; % B^{-1} explicite
X = [x.^2+y.^2, x, y, ones(size(x))];
[Q R] = qr(X,0);
[V L] = eig(R*invB*R');
[L i] = sort(diag(L));
c = R \ V(:,i(2)); % interesuje nas druga najmniejsza w.wl, a raczej: wektor
S = [-c(2:3)/(2*c(1)); sqrt(c(2)^2 + c(3)^2 - 4*c(1)*c(4))/(2*abs(c(1)))];
dist = residgeom(S,x,y);
info = 1;
end
```

**Ćwiczenie 7.1.** Przeredaguj funkcje `fitcircle1` i `fitcircle2` tak, by jeśli użytkownik nie poda przybliżenia startowego, zostało wybrane przybliżenie wyznaczanie funkcją `fitcircle3`.

*Rozwiązanie.* Musimy zmienić kolejność argumentów tak, by przybliżenie początkowe było ostatnim. Wtedy początek funkcji, np. `fitcircle1`, należałoby zapisać np. w takiej formie:

```
function [S,dist,info] = fitcircle1(x,y,S0)
if nargin < 3
    S0 = fitcircle3(x,y);
end
% ... tu dalsze instrukcje ...
end
```

Poniżej przytaczamy przykładowy skrypt dopasowujący okrąg do zadanych punktów trzema opisanymi wcześniej metodami.

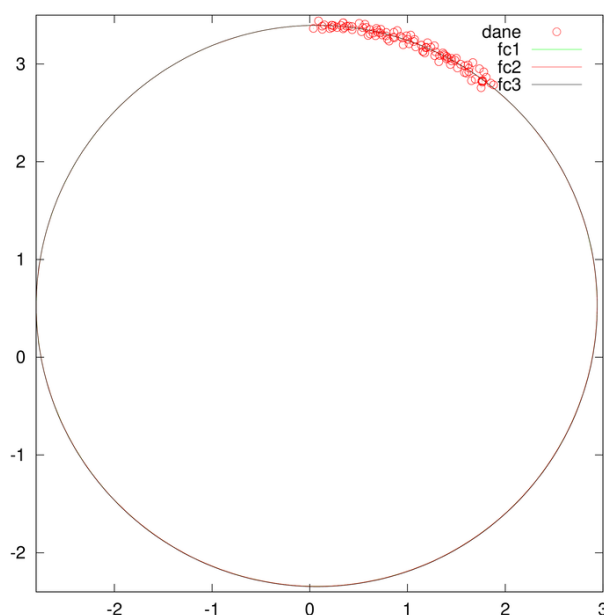
```

t = linspace(0,0.2*pi,100)';
X = 3*sin(t)+0.2*rand(size(t));
Y = 3.35*cos(t)+0.1*rand(size(t));

XY = [X,Y];
e = fitcircle3(X,Y);
c = fitcircle1(e,X,Y);
d = fitcircle2(e,X,Y);
[e c d]

t = linspace(0,2*pi,300);
plot(X,Y,'ro');
axis('square');
hold on;
plot(c(1) + c(3)*sin(t),c(2)+c(3)*cos(t),'g-');
plot(d(1) + d(3)*sin(t),d(2)+d(3)*cos(t),'r-');
plot(e(1) + e(3)*sin(t),e(2)+e(3)*cos(t),'k-');
plot(X,Y,'ro');
hold off;
legend('dane', 'fc1', 'fc2', 'fc3');
[residgeom(c,X,Y), residgeom(d,X,Y), residgeom(e,X,Y)]

```



Rysunek 7.2. Przykładowy wynik dopasowania okręgu trzema opisywanymi metodami, gdy zadane punkty prawie leżą na pewnym okręgu. W tym (typowym) przypadku, każda z metod daje praktycznie identyczny wynik!.

### 7.3. Uwagi i uzupełnienia

W przypadku, gdy wiadomo, które punkty  $P$  należą do jakich boków prostokąta, zadanie geometrycznego dopasowania prostokąta jest w miarę prostym uogólnieniem zadania dopasowa-



---

nia linii, zob [7, Chapter 6, str. 88]. [1] jest prawdziwą kopalnią wiedzy na temat dopasowywania prostych, okręgów i elips.

## 8. Stacjonarne równanie dyfuzji

W bardzo wielu praktycznych modelach (biologicznych, chemicznych, fizycznych i nawet ekonomicznych) należy rozwiązać równanie różniczkowe postaci

$$-\operatorname{div}(a(x)\nabla u(x)) = f(x), \quad x \in \Omega, \quad (8.1)$$

$$u(x) = g(x) \quad x \in \partial\Omega. \quad (8.2)$$

Niewiadomą jest funkcja  $u : \bar{\Omega} \rightarrow \mathbb{R}$ . W klasycznym modelu fizycznym dyfuzji,  $u$  odpowiadałoby stężeniu jakiejś substancji, a współczynnik  $a$  nazywalibyśmy współczynnikiem dyfuzji. Zadana funkcja  $f$  byłaby interpretowana jako zewnętrzne źródło substancji, a warunek brzegowy typu Dirichleta oznaczałby, że na brzegu obszaru utrzymujemy zadane stężenie równe  $g$ . Warto pamiętać, że równaniem tego samego lub podobnego typu można opisywać wiele innych sytuacji, na przykład rozchodzenie się ciepła w pewnym materiale.

Ze względu na powszechność występowania takich i podobnych modeli w zastosowaniach matematyki, trudno wyobrazić sobie, by absolwent matematyki stosowanej nie potrafił skutecznie zmierzyć się z takim zadaniem, choćby w *najprostszej* postaci — wykorzystując w tym celu choćby *najprostsze* metody. Dlatego w niniejszym rozdziale zajmiemy się szczegółowo rozwiązaniem takiego zadania w Octave.

W ogólnym przypadku, moglibyśmy mieć do czynienia z macierzą współczynników  $a(x)$ , ale tutaj dla uproszczenia (zadanie i tak będzie obliczeniowo wystarczająco skomplikowane) przyjmiemy, że

- Współczynnik dyfuzji jest dodatnią funkcją skalarną,  $a : \bar{\Omega} \rightarrow \mathbb{R}_+$  i ograniczoną: istnieją stałe absolutne  $m, M$  takie, że  $0 < m \leq |a(\cdot)| \leq M$ .
- $\Omega$  jest kostką w  $\mathbb{R}^d$ , przy czym  $d \in \{1, 2, 3\}$  — czyli rozpatrujemy dyfuzję w co najwyżej trójwymiarowym obszarze o bardzo prostej geometrii. Co robić w przypadku, gdy  $\Omega$  nie jest kostką, powiemy parę słów pod koniec rozdziału.

Zacniemy więc od samodzielnej dyskretyzacji równania (8.1) w najprostszym przypadku.

### 8.1. Laplasjan: dyskretyzacja metodą różnic skończonych

Na początek przyjmijmy, że warunek brzegowy jest jednorodny:  $g = 0$  oraz  $a(x) \equiv 1$ . Wówczas zadanie (8.1) upraszcza się do równania Poissona:

$$-\Delta u(x) = f(x) \quad x \in \Omega, \quad (8.3)$$

$$u(x) = 0 \quad x \in \partial\Omega. \quad (8.4)$$

Najprostszą, choć zazwyczaj najmniej skuteczną (przez swój brak wyrafinowania i dostosowania do faktycznego przebiegu rozwiązania) metodą dyskretyzacji takiego równania jest użycie *równomiernej* siatki na  $\Omega$  i metody różnic skończonych.

### 8.1.1. Dyskretyzacja jednowymiarowego laplasjanu

Gdy  $d = 1$ , to  $\Omega$  jest odcinkiem<sup>1</sup>,  $\Omega = (0, X)$ , a (8.3) staje się równaniem różniczkowym zwyczajnym z warunkiem brzegowym:

$$-\frac{d^2}{dx^2}u(x) = f(x) \quad \forall x \in (0, X), \quad (8.5)$$

$$u(0) = u(X) = 0. \quad (8.6)$$

Aby znaleźć jego przybliżone rozwiązanie, możemy na przykład zastąpić równanie różniczkowe odpowiadającym mu równaniem *różnicowym*,

$$-\frac{u_{i-1} - 2u_i + u_{i+1}}{h_x^2} = f_i \quad \forall i = 1, \dots, N_x, \quad (8.7)$$

$$u_0 = u_{N_x+1} = 0, \quad (8.8)$$

gdzie  $u_i$  ma odpowiadać wartości rozwiązania w węźle dyskretyzacji  $x_i$ ,  $u_i \approx u(x_i)$ , oraz  $x_i = i \cdot h_x$ , przy czym  $h_x = X/(N_x + 1)$ . Odpowiedzią na pytanie, jak dobrze (i czy w ogóle) takie rozwiązanie *dyskretnie* aproksymuje rozwiązanie dokładne, zajmujemy się na wykładzie z [Numerycznych równań różniczkowych](#); tutaj interesuje nas przede wszystkim aspekt praktyczny, czyli metoda uzyskania konkretnego przybliżenia. Najpierw zobaczmy więc, do jakiego zagadnienia algebraicznego prowadzi równanie różnicowe (8.7).

Jest to oczywiście równanie liniowe na współczynniki  $U_{N_x} = (u_1, \dots, u_{N_x})^T$ ,

$$T_{N_x} U_{N_x} = F_{N_x},$$

gdzie

$$T_{N_x} = \frac{1}{h_x^2} \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}_{N_x \times N_x}. \quad (8.9)$$

Ponieważ interesują nas dobre przybliżenia (to znaczy przypadek, gdy  $h_x$  jest małe — lub *bardzo* małe), znaczy to, że  $N_x$  będzie (bardzo) duże. Nasza macierz jest macierzą rozrzedzoną, bo ma nieco mniej niż  $3N_x$  niezerowych elementów. Jej współczynnik wypełnienia (*density*), czyli stosunek liczby niezerowych do liczby wszystkich elementów macierzy, rzędu  $1/N_x$ , maleje ze wzrostem  $N_x$ . Nie ma więc sensu pamiętać *wszystkich*  $N_x^2$  jej elementów — raczej, pamiętalibyśmy tylko jej niezerowe elementy. Do tego świetnie nadaje się format macierzy rozrzedzonych Octave. Dowolną macierz w takim formacie można utworzyć poleceniem **sparse**, ale ze względu na to, że nasza konkretna macierz ma wyjątkowo prostą strukturę, użyjemy polecenia

```
hx = X/(Nx+1);
Tx = spdiags( repmat([-1 2 -1]/(hx^2), Nx, 1), [-1,0,1], Nx, Nx);
```

gdzie wcześniej musimy zdefiniować  $X = X$   $N_x = N_x$ . Komenda **repmat** klonuje daną macierz  $N_x$  razy.

Dzięki reprezentacji naszej macierzy w formacie rzadkim, macierz rozmiaru  $N_x$  zajmie nam w pamięci tylko około  $8 \cdot 3 \cdot N_x = 24 N_x$  bajtów (w Octave wszystkie liczby rzeczywiste są domyślnie reprezentowane w podwójnej precyzji, a więc na 64 bitach, czyli 8 bajtach) — a nie standardowe  $8 N_x^2$  bajtów w przypadku, gdybyśmy reprezentowali ją jako macierz pełną.

<sup>1</sup> Bez zmniejszenia ogólności założymy, że początek odcinka znajduje się w punkcie  $x = 0$ .

**Ćwiczenie 8.1.** Spróbuj zaimplementować obliczanie  $T_x$ , zastępując w kodzie powyżej

```
[-1 2 -1]/(hx^2)
```

przez pozornie „równie dobrze wyglądające”

```
[-1 2 -1]/hx*hx
```

Czy dostajesz tę samą macierz  $T_x$ ? To jeszcze jeden przyczynek do tego, że ważne jest, by *na różne sposoby* zweryfikować poprawność konkretnej implementacji.

### 8.1.2. Rozwiązanie równania Poissona w obszarze jednowymiarowym

Otrzymana powyżej macierz jest trójdzielna (a przy tym: symetryczna i dodatnio określona), zatem właściwie zrealizowany algorytm eliminacji Gaussa (lub lepiej: rozkładu  $LDL^T$ ) pozwoli nam wyznaczyć rozwiązanie układu  $T_{N_x} U_{N_x} = F_{N_x}$  kosztem  $O(N_x)$  flopów — a więc, z dokładnością do stałej, optymalnym. Standardowy operator rozwiązywania równań w Octave — „\” — sprawdza na wstępie, czy macierz układu jest właśnie rozrzedzona, a jeśli tak — stosuje odpowiedni *solver* oparty na eliminacji Gaussa, w naszym przypadku w pełni uwzględniający trójdzielną strukturę. Dlatego rozwiązanie naszego układu skutecznie i szybko obliczymy komendą

```
Ux = Tx \ Fx;
```

**Ćwiczenie 8.2.** Wyznacz przybliżone rozwiązanie równania

$$-k \frac{d^2}{dx^2} u(x) = f(x), \quad x \in (0, 2),$$

$$u(0) = u(2) = 0,$$

gdzie  $k = 2.45$  oraz

$$f(x) = \begin{cases} 1/(2-x), & x \in (0, 1], \\ 1/x^5, & x \in [1, 2). \end{cases}$$

*Rozwiązanie.* Dzieląc równanie przez stałą  $k$  dostajemy (8.20) dla prawej strony równej  $f(x)/k$ . Ponieważ  $f(x)$  załamuje się w  $x = 1$ , weźmiemy siatkę, w której jeden z węzłów wypadnie akurat w  $x = 1$ . W przypadku siatki jednorodnej zagwarantujemy to sobie, biorąc nieparzyste  $N_x$ . Wstępnie wydaje się, że biorąc  $N_x = 99$   $X + 1 = 199$ , uzyskamy dostatecznie dużą rozdzielczość schematu (wszak wówczas  $h_x = 10^{-2}$ ). Potem postaramy się sprawdzić inżynierską metodą, jaka jest jakość obliczonego rozwiązania.

Najpierw musimy więc zdefiniować podstawowe obiekty wykorzystywane w procesie dyskretyzacji.

```
X = 2; % fizyczne rozmiary obszaru
k = 2.45; % stała dyfuzji
```

```
Nx = 199;
hx = X/(Nx+1);
px = hx*[0:Nx+1]'; % prawdziwe współrzędne węzłów dyskretyzacji
    % potrzebne m.in. do definicji prawej strony;
    % określone łącznie z brzegiem obszaru
Tx = lap1ddset(Nx,hx); % macierz laplasjanu
```

```
% wyznaczamy funkcję prawej strony
```

```

m = 1; % punkt graniczny
pxl = px(px<m); pxr = px(px>=m); % węzły na lewo i prawo od "m"
Fx = [1./(2-pxl); 1./(pxr.^5)] / k;
plot(px,Fx); % sprawdzamy wizualnie poprawność Fx
Fx = Fx(2:end-1); % wybieramy tylko wartości w wewnętrznych węzłach

```

Zwróćmy uwagę na sposób generowania prawej strony: ponieważ musimy zdefiniować ją jedynie w wewnętrznych węzłach siatki, odwołujemy się jedynie do węzłów  $F_x(2:\text{end}-1)$ . Dla większej skuteczności (i przejrzystości) kodu użyliśmy operatora potęgowania tablicowego. Do zdefiniowania macierzy  $T_{N_x}$  wykorzystaliśmy funkcję pomocniczą `lap1ddset`,

```

function T = lap1ddset(N, h)
    T = spdiags( repmat([-1 2 -1]./(h^2), N, 1), [-1,0,1], N, N);
end

```

Teraz wystarczy rozwiązać równanie i wizualizować wynik:

```

Ux = Tx \ Fx;
plot(px, [0; Ux; 0]); legend('U_{N_x}(x)'); grid on;

```

Pozostaje przekonać się, czy uzyskane przez nas rozwiązanie jest wystarczająco dokładne. W tym celu zastosujemy inżynierski sposób: sprawdzimy, jak bardzo różni się nasze rozwiązanie od rozwiązania uzyskanego na *gęstszej* siatce. Zapamiętamy więc  $U_x$  pod tymczasową nazwą,

```
TUx = Ux;
```

i ponownie uruchomimy symulację, startując jednak z  $N_x = 399$  (a więc, używając dwa razy drobniejszej siatki). Wtedy możemy „oszacować” popełniony błąd:

```

octave:12> norm(Ux(2:2:end-1)-TUx,Inf)
ans = 9.5256e-06

```

Zatem rozwiązanie uzyskane na dwukrotnie zagęszczonej siatce (a więc zapewne dokładniejsze, bo błąd aproksymacji maleje jak  $O(h^2)$ , por. wykład z [numerycznych równań różniczkowych](#), rozdział dotyczący metody różnic skończonych dla równań eliptycznych) różni się w punktach starej siatki od oryginalnie uzyskanego jedynie o około  $10^{-5}$ . To oczywiście tylko jest pewna sugestia co do błędu realnego, gdyż na pewno możemy powiedzieć jedynie, że w dyskretnej normie maksimum określonej w węzłach rzadszej siatki,

$$\|u - U_{N_x}\|_\infty \leq \underbrace{\|U_{N_x} - U_{2N_x+1}\|_\infty}_{\approx 10^{-5} \text{ (?)}} + \underbrace{\|u - U_{2N_x+1}\|_\infty}_{\text{wierzymy, że jest „małe” wobec poprzedniego}}.$$

## Testy

Nasza praca nie będzie zakończona, dopóki nie zweryfikujemy na kilku testowych przykładach, czy wykorzystana metoda produkuje rezultaty zgodne z oczekiwaniami. Minimalny zestaw testów mógłby tutaj wyglądać następująco:

- Czy  $T_{N_x}$  jest symetryczna? Czy jest trójdzielna tak, jak chcieliśmy?
- Czy  $h_x^2 T_{N_x}$  jest macierzą o diagonalu równej 2?
- Ile wynosi  $T_{N_x} U_{N_x}$ , gdy  $U_{N_x}$  jest wektorem stałym? (Wynik powinien być zerowy z wyjątkiem pierwszego i ostatniego węzła.)
- Czy jeśli  $F_{N_x} = 2$ , to rozwiązując zadanie dyskretne dostajemy w wyniku dyskretyzację funkcji  $x(X - x)$ ? A dla  $F_{N_x} = 0$ ?

- Funkcja  $u(x) = \sin\left(\frac{2\pi}{X}x\right)$  spełnia równanie Poissona dla  $f(x) = \left(\frac{2\pi}{X}\right)^2 u(x)$ . Należy zbadać, czy błąd rozwiązań dyskretnych uzyskiwanych dla siatek o oczku  $h_x, h_x/2, h_x/4, \dots$ , maleje zgodnie z teoretycznymi przewidywaniami dla tego zadania, por. wykład z [numerycznych równań różniczkowych](#), rozdział o metodzie różnic skończonych dla równań eliptycznych.

Wykonanie tych testów pozostawiamy Czytelnikowi.

**Ćwiczenie 8.3.** Przeprowadź testy poprawności implementacji zarówno macierzy  $T_{N_x}$ , jak i sposobu rozwiązywania jednowymiarowego równania Poissona.

*Wskazówka.* Aby na przykład sprawdzić wynik działania operatora  $T_{N_x}$  na wektorze stałym, wydamy komendę

```
Ux = ones(size(Fx));
s = Tx*Ux;
norm(s(2:end-1),Inf)
plot(px,[0;s;0]); grid on;
```

Obliczona powyżej norma fragmentu  $s$  powinna być równa zero, a wykres powinien mieć dwa piki w punktach tuż przy krańcach odcinka. Oczywiście, nie jest głupim pomysłem sprawdzenie wyniku dla kilku (np. małej i dużej) wartości  $N_x$ .

### 8.1.3. Dyskretyzacja dwuwymiarowego laplasjanu

Chcąc rozwiązać dwuwymiarowe zadanie Poissona,

$$-\frac{\partial^2}{\partial x^2}u(x,y) - \frac{\partial^2}{\partial y^2}u(x,y) = f(x,y) \quad \forall (x,y) \in (0,X) \times (0,Y) = \Omega, \quad (8.10)$$

$$u(x,y) = 0 \quad \forall (x,y) \in \partial\Omega, \quad (8.11)$$

postąpimy analogicznie jak w poprzednim przypadku, korzystając z jednorodnej siatki węzłów na prostokącie  $\Omega$ .

Będziemy zatem mieli do czynienia z wartościami rozwiązania w punktach  $(x_i, y_j)$ , gdzie

$$x_i = i \cdot h_x, \quad \text{dla } i = 0, \dots, N_x + 1, \quad \text{oraz} \quad y_j = j \cdot h_y, \quad \text{dla } j = 0, \dots, N_y + 1,$$

przy czym

$$h_x = \frac{X}{N_x + 1} \quad \text{oraz} \quad h_y = \frac{Y}{N_y + 1}.$$

Wartości rozwiązania dyskretnego  $u_{i,j}$  będą odpowiadać wartościom rozwiązania w węzłach dyskretyzacji  $(x_i, y_j)$ , tzn.  $u_{i,j} \approx u(x_i, y_j)$ .

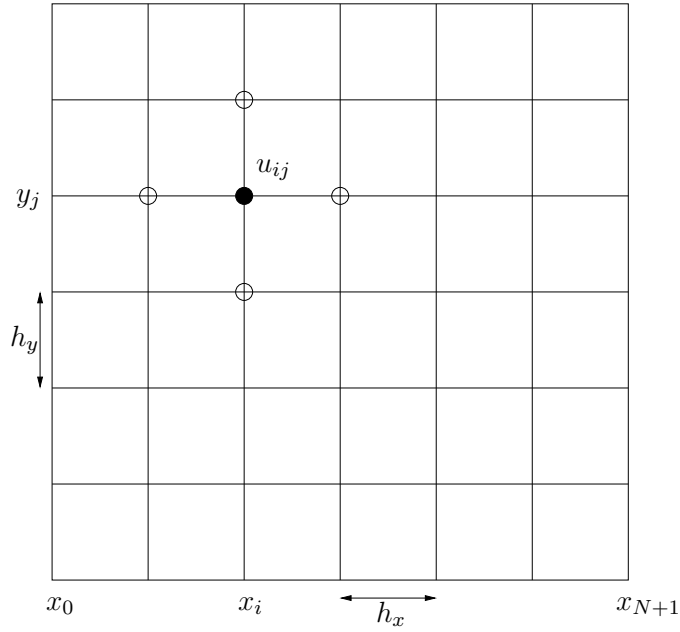
Aby znaleźć przybliżone rozwiązanie (8.10), równanie różniczkowe zastąpimy odpowiadającym mu równaniem różnicowym, korzystając z tego samego co poprzednio sposobu aproksymacji drugiej pochodnej:

$$\frac{\partial^2}{\partial x^2}u(x_i, y_j) \approx \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h_x^2}$$

i analogicznie dla  $\frac{\partial^2}{\partial y^2}(x_i, y_j)$ . Otrzymamy więc w końcu równanie różnicowe postaci

$$-\frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h_x^2} - \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{h_y^2} = f_{i,j} \quad \forall i = 1, \dots, N_x, j = 1, \dots, N_y, \quad (8.12)$$

$$u_{0,j} = u_{N_x+1,j} = u_{i,0} = u_{i,N_y+1} = 0, \quad \forall i = 0, \dots, N_x + 1, j = 0, \dots, N_y + 1, \quad (8.13)$$

Rysunek 8.1. Regularna siatka na  $\Omega$ .

gdzie  $f_{i,j} = f(x_i, y_j)$ . Mamy więc do wyznaczenia  $N = N_x \cdot N_y$  niewiadomych, które w naturalny sposób układają się w macierz

$$\begin{pmatrix} u_{1,1} & u_{1,2} & u_{1,3} & \dots & u_{1,N_y} \\ u_{2,1} & u_{2,2} & u_{2,3} & \dots & u_{2,N_y} \\ \vdots & & & & \\ u_{N_x,1} & u_{N_x,2} & u_{N_x,3} & \dots & u_{N_x,N_y} \end{pmatrix}$$

(zwróćmy uwagę na to, że „fizyczne” współrzędne są odwrócone: wartości odpowiadające kolejnym współrzędnym w kierunku osi  $OX$  znajdują się w kolejnych *wierszach* jednej *kolumny* powyższej macierzy!). Podobnie jest z wartościami prawej strony równania. Aby jednak korzystać ze standardowych narzędzi Octave (na przykład operatora „\” rozwiązywania układów równań liniowych), nasze zadanie musimy sformułować w postaci, w której niewiadome (a także wektor prawej strony) są reprezentowane w postaci *wektora*, długości  $N$ . Jakkolwiek można to robić na wiele sposobów, nam będzie najwygodniej uporządkować elementy kolumnami<sup>2</sup>:

$$U_N = (\underbrace{u_{1,1}, u_{2,1}, u_{3,1}, \dots, u_{N_x,1}}_{\text{niewiadome pierwszej kolumny}}, \underbrace{u_{1,2}, u_{2,2}, u_{3,2}, \dots, u_{N_x,2}}_{\text{drugiej kolumny}}, \dots, \underbrace{u_{1,N_y}, u_{2,N_y}, u_{3,N_y}, \dots, u_{N_x,N_y}}_{\text{ostatniej kolumny}})^T. \quad (8.14)$$

Analogicznie zdefiniowalibyśmy także wektor prawej strony  $F_N$ . Będziemy więc szukać wektora  $U_N$ , który spełnia liniowy układ równań o macierzowej postaci

$$P_N U_N = F_N,$$

<sup>2</sup> Ze względu na to, że łatwo to osiągnąć w Octave; patrz niżej.

gdzie  $P_N$  jest pięciodiagonalną macierzą o blokowej strukturze,

$$P_N = \begin{pmatrix} T_{N_x} + \frac{2}{h_y^2}I & -\frac{1}{h_y^2}I & & & \\ -\frac{1}{h_y^2}I & T_{N_x} + \frac{2}{h_y^2}I & -\frac{1}{h_y^2}I & & \\ & \ddots & \ddots & \ddots & \\ & & -\frac{1}{h_y^2}I & T_{N_x} + \frac{2}{h_y^2}I & -\frac{1}{h_y^2}I \\ & & & -\frac{1}{h_y^2}I & T_{N_x} + \frac{2}{h_y^2}I \end{pmatrix}_{N \times N}. \quad (8.15)$$

W kolejnych diagonalnych blokach  $P_N$  znajdują się macierze trójdzielne  $T_{N_x} + \frac{2}{h_y^2}I$ , gdzie  $T_{N_x}$  jest znaną już nam macierzą dyskretyzacji jednowymiarowego laplasjanu. Jest tych bloków  $N_y$ . Używając symbolu Kroneckera dla iloczynu tensorowego macierzy mamy po prostu

$$P_N = I_{N_y} \otimes T_{N_x} + T_{N_y} \otimes I_{N_x}.$$

Przypomnijmy na marginesie, że iloczyn tensorowy macierzy  $A$  i  $B$  jest macierzą o strukturze blokowej, postaci

$$\begin{pmatrix} a_{11}B & a_{12}B & \dots & a_{1M}B \\ a_{21}B & a_{22}B & \dots & a_{2M}B \\ \vdots & \vdots & \dots & \\ a_{N1}B & a_{N2}B & \dots & a_{NM}B \end{pmatrix}.$$

W Octave istnieje funkcja **kron**, która wyznacza iloczyn Kroneckera macierzy, przy czym zastosowana do macierzy rozrzedzonej daje w rezultacie także macierz rozrzedzoną. Dlatego definicja macierzy  $P_N$  będzie w Octave wyjątkowo krótka:

```
hx = X/(Nx+1); hy = Y/(Ny+1);
Tx = lap1ddset(Nx, hx); lx = speye(Nx, Nx);
Ty = lap1ddset(Ny, hy); ly = speye(Ny, Ny);
P = kron(ly, Tx) + kron(Ty, lx);
```

**Ćwiczenie 8.4.** Zweryfikuj poprawność swojej implementacji macierzy  $P_N$ .

#### 8.1.4. Rozwiązanie równania Poissona w obszarze dwuwymiarowym

W przypadku macierzy  $P_N$ , prosta strategia polegająca na eliminacji Gaussa (z wykorzystaniem dodatkowo faktu, że  $P_N$  jest pasmowa, o szerokości pasma  $2N_x$ ), dałaby nam szansę rozwiązać to równanie kosztem  $O(N_x^2 N_y^2)$ , co jest, przy dużych wielkościach  $N_x$  i  $N_y$ , wartością trudną do zaakceptowania. Na szczęście, wbudowany Octave *solver* układów równań liniowych z macierzami rzadkimi, kryjący się za operatorem „\”, potrafi (czasem!) uporządkować niewiadome i równania tak, żeby wyraźnie obniżyć koszt rozwiązania.

Zatem, moglibyśmy w dalszym ciągu wyznaczać rozwiązanie poleceniem

```
U = P \ F;
```

o ile tylko  $F_N$  w odpowiedni sposób przedstawimy w postaci długiego wektora  $F$ . Uzyskane rozwiązanie,  $U$ , będzie dla nas z pewnością bardziej użyteczne, jeśli „zwiniemy” je z powrotem do postaci macierzowej:  $U = \text{reshape}(U, N_x, N_y)$ .

Jak to powinno wyglądać w szczegółach, jest treścią poniższego ćwiczenia.



**Ćwiczenie 8.5.** Wyznacz przybliżone rozwiązanie równania dyfuzji w środowisku anizotropowym,

$$\begin{aligned} \frac{\partial^2}{\partial x^2} u(x, y) - k \frac{\partial^2}{\partial y^2} u(x, y) &= f(x, y), & (x, y) \in (0, 2) \times (0, 3) = \Omega, \\ u(x, y) &= 0, & (x, y) \in \partial\Omega, \end{aligned}$$

gdzie  $k = 2.45$  oraz

$$f(x, y) = \begin{cases} 1, & |x - 1| \leq \frac{1}{2}, |y - 2| \leq \frac{1}{2}, \\ 0, & \text{w przeciwnym przypadku.} \end{cases}$$

Narysuj wykres  $u(x, y)$  i zbadaj następnie, jak rozwiązanie zmienia się wraz ze zmianą  $k$ .

*Rozwiązanie.* Zgodnie z wyżej opisanym schematem, musimy wyznaczyć macierz  $P_N$  odpowiadającą równaniu różnicowemu

$$-\frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h_x^2} - k \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{h_y^2} = f_{i,j}$$

(zwróćmy uwagę na obecność współczynnika  $k$  przy dyskretyzacji drugiej pochodnej względem  $y$ ). Zatem, postępując zgodnie z opisaną metodą, po dyskretyzacji równania dostaniemy macierz

$$P_N = I_{N_y} \otimes T_{N_x} + (k T_{N_y}) \otimes I_{N_x}.$$

Reszta jest już tylko (?) programowaniem. Musimy przy tym pamiętać, by we właściwym momencie z dwuwymiarowej tablicy wartości rozwiązania/prawej strony przechodzić do postaci rozwiniętej w wektor (lub na odwrót).

`X = 2; Y = 3; k = 2.45;`

`Nx = 100; Ny = 200;`

`hx = X/(Nx+1); hy = Y/(Ny+1);`

`Tx = lap1ddset(Nx, hx); lx = speye(Nx, Nx);`

`Ty = k*lap1ddset(Ny, hy); ly = speye(Ny, Ny);`

`P = kron(ly, Tx) + kron(Ty, lx);`

**Chwila refleksji** Jest jeszcze jeden ważny szczegół: funkcja prawej strony *nie jest* ciągła, zatem dokładne rozwiązanie nie może być klasy  $C^2(\Omega)$  — nie będzie klasyczne. Owszem, możemy spodziewać się rozwiązań słabych — klasy  $H_0^1(\Omega)$  — ale czy uprawnione w tym przypadku jest używanie aproksymacji różnicowej dla *drugiej* pochodnej, opartej na rozwinięciach Taylora?

Ten dylemat natury teoretycznej wydaje się być nieistotnym dla praktyka — w końcu przecież wystarczy użyć naszego sposobu i sprawdzić, „czy sobie poradzi”. Jednak kłopot polega na tym, że nie zawsze mamy wystarczające podstawy, by rozstrzygnąć, czy otrzymane rozwiązanie jest „sensowne”. Użycie kryterium czysto estetycznego („ładnie wygląda”) niestety często prowadzi do pochopnej akceptacji całkowicie fałszywych wyników.

Zatem wstępne zastanowienie się nad jakością używanych narzędzi aproksymacji problemu nie jest bez znaczenia, bo użycie błędnej/niewłaściwej dyskretyzacji może w efekcie doprowadzić nas do efektownych wizualizacji, które nie będą miały wiele wspólnego z realnym rozwiązaniem. Zachęcając Czytelnika do poszukania odpowiedzi<sup>3</sup> na to pytanie na przykład w naszym

<sup>3</sup> Okazuje się, że nasza dyskretyzacja ma sens nawet w tym przypadku: gdy potraktować ją jako dyskretyzację metodą elementu skończonego z nienajlepszą (ale wciąż sensowną) aproksymacją wektora prawej strony. Następnym etapem rozwiązywania tego zadania byłoby więc „podrasowanie” wektora prawej strony.

ulubionym rozdziale o metodzie różnic skończonych dla równań eliptycznych w wykładzie z [numerycznych równań różniczkowych](#), teraz — *na własne ryzyko* — (a przy tym dla lepszego rozeznania) sprawdzimy jednak po prostu, „co wyjdzie”.

Na początek wygodnie będzie nam utworzyć macierz  $F = (F_{i,j})$  określoną we wszystkich węzłach dyskretyzacji (łącznie z brzegowymi):

```
px = hx*[0:Nx+1]'; % prawdziwe współrzędne węzłów dyskretyzacji
py = hy*[0:Ny+1]';
[pX, pY] = meshgrid(px,py); % tablica węzłów do manipulacji tablicowej

F = zeros(size(pX));
[i,j] = find( (abs(pX-1) <= 0.5) & (abs(pY-2) <= 0.5) );
F(i,j) = 1;
F = permute(F, [2 1]); % zamienia ze sobą wiersze i kolumny
```

Ostatni krok może wydać się nam niejasny: dlaczego musimy zamienić ze sobą wiersze i kolumny macierzy  $F$ ? Przyczyną jest struktura, jaką mają macierze  $pX$ ,  $pY$ , które Octave generuje poleceniem **meshgrid**. Aby ułatwić wizualizację takich macierzy, indeksy odpowiadające zmiennej  $x$  odpowiadają tam numerom *kolumn* — a nie, tak, jak my sobie to założyliśmy, wierszy. Określając więc wartości  $F$  na podstawie  $pX$  i  $pY$ , dostajemy  $F$  o zamienionych ze sobą wierszach i kolumnach. Dlatego na koniec musimy to odwrócić poleceniem **permute**<sup>4</sup>.

Chcąc teraz wizualizować macierz  $F$ , musimy ponownie zamienić ze sobą wymiary: polecenie **imagesc** oczekuje macierzy wartości w węzłach takiej, jak np. produkowana poleceniem **meshgrid**.

```
imagesc(px, py, permute(F, [2 1]));
axis('xy','square');
title('Prawa strona');
```

Na zakończenie musimy z tablicy  $F$  usunąć niepotrzebne wartości odpowiadające węzłom leżącym na brzegu, a następnie „rozprostować”  $F$  do wektora:

```
F = F(2:Nx+1,2:Ny+1);
F = F(:);
```

Ostatnie polecenie tworzy wektor, którego elementy są kolejnymi elementami kolejnych *kolumn* macierzy  $F$  — dokładnie tak, jak umawialiśmy się w (8.14).

Teraz wystarczy zadanie rozwiązać, a wynik na powrót *zwinąć* do tablicy i wizualizować.

```
U = zeros(Nx+2,Ny+2);
U(2:Nx+1, 2:Ny+1) = reshape(F, Nx, Ny);
imagesc(px, py, permute(U, [2 1]));
axis('xy','square');
title('Rozwiazanie');
```

Na zakończenie powinniśmy ocenić (w analogiczny sposób jak w przypadku jednowymiarowym), jakiego rzędu błąd popełniliśmy stosując naszą dyskretyzację. Wykonanie odpowiednich testów pozostawiamy Czytelnikowi.

<sup>4</sup> Tak, moglibyśmy też użyć transpozycji, ale nie uogólniłoby się to na przypadek trójwymiarowy, którym za niedługo będziemy się zajmować.

### 8.1.5. Dyskretyzacja i rozwiązanie równania Poissona w obszarze trójwymiarowym

Czytelnik zechce sam sprawdzić, że dyskretyzacja operatora Laplace’a w kostce  $\Omega = (0, X) \times (0, Y) \times (0, Z)$  różnicami skończonymi na regularnej siatce o oczku  $(h_x, h_y, h_z)$  i  $N_x \times N_y \times N_z$  wewnętrznych węzłach, prowadzi do zadania różnicowego

$$-\frac{u_{i-1,j,k} - 2u_{i,j,k} + u_{i+1,j,k}}{h_x^2} - \frac{u_{i,j-1,k} - 2u_{i,j,k} + u_{i,j+1,k}}{h_y^2} - \frac{u_{i,j,k-1} - 2u_{i,j,k} + u_{i,j,k+1}}{h_z^2} = f_{i,j,k} \quad (8.16)$$

dla  $i = 1, \dots, N_x$ ,  $j = 1, \dots, N_y$ ,  $k = 1, \dots, N_z$ , z odpowiednimi warunkami brzegowymi. Oczywiście  $u_{i,j,k} \approx u(ih_x, jh_y, kh_z)$ . Mamy więc do wyznaczenia  $N = N_x \cdot N_y \cdot N_z$  niewiadomych wewnątrz obszaru, które w naturalny sposób układają się w *trójwymiarową* tablicę

$$(u_{i,j,k})_{N_x \times N_y \times N_z}.$$

Aby wciąż korzystać ze standardowych narzędzi Octave jak w przypadkach dwu- i jednowymiarowym, nasze zadanie musimy sformułować w postaci, w której niewiadome tworzą jeden wektor długości  $N$ . Z objaśnionych wcześniej powodów, zrobimy to „kolumnami” — to znaczy tak, by w uzyskanym wektorze najszybciej zmieniała się pierwsza współrzędna, potem druga, a najwolniej — trzecia:

$$U_N = \left( \underbrace{u_{1,1}, u_{2,1,1}, u_{3,1,1}, \dots, u_{N_x,1,1}}_{\text{niewiadome pierwszej kolumny i pierwszego plastra}}, \underbrace{u_{1,2,1}, u_{2,2,1}, u_{3,2,1}, \dots, u_{N_x,2,1}, \dots}_{\text{drugiej kolumny i pierwszego plastra}}, \underbrace{u_{1,N_y,1}, u_{2,N_y,1}, u_{3,N_y,1}, \dots, u_{N_x,N_y,1}, \dots}_{\text{ostatniej kolumny i pierwszego plastra}}, \dots, \underbrace{u_{1,N_y,N_z}, u_{2,N_y,N_z}, u_{3,N_y,N_z}, \dots, u_{N_x,N_y,N_z}}_{\text{ostatniej kolumny i ostatniego plastra}} \right)^T.$$

Będziemy szukać wektora  $U_N$ , który spełnia liniowy układ równań

$$S_N U_N = F_N,$$

gdzie  $S_N$  jest siedmiodiagonalną macierzą o blokowej strukturze,

$$S_N = I_{N_z} \otimes I_{N_y} \otimes T_{N_x} + I_{N_z} \otimes T_{N_y} \otimes I_{N_x} + T_{N_z} \otimes I_{N_y} \otimes I_{N_x}. \quad (8.17)$$

Implementacja takiej macierzy, a następnie jej rozwiązanie za pomocą standardowego bezpośredniego *solwera* z Octave przebiega w sposób w pełni analogiczny do przypadku dwuwymiarowego. Niestety, gdy ledwie dziesięciokrotnie zmniejszymy rozmiar oczka siatki w każdym kierunku, liczba niewiadomych rośnie aż tysiąckrotnie. Otrzymane zadanie będzie więc bardzo duże: faktycznie, już dla umiarkowanego  $N_x = N_y = N_z = 100$  dochodzimy do  $N = 10^6$  (!) Dodatkowo, struktura naszej (bardzo rozrzedzonej) macierzy jest na tyle skomplikowana dla bezpośredniego *solwera* opartego na eliminacji Gaussa, że szybko staje się on nieskuteczny.

**Ćwiczenie 8.6.** Rozwiąż zagadnienie Poissona

$$\begin{aligned} -\Delta u &= f & \text{w } \Omega &= (0, 1) \times (0, 2) \times (0, 3), \\ u &= 0 & \text{na } \partial\Omega, \end{aligned}$$

gdzie  $f$  przyjmuje wartość 1, zaburzona o losowe wartości z przedziału  $[-0.1, 0.1]$ . Wizualizuj rozwiązanie.

*Rozwiązanie.* Tak sformułowane zadanie ma charakter czysto jakościowy, a nie ilościowy — rzeczywiście, skoro  $f$  jest poddana losowemu zaburzeniu, to trudno spodziewać się, by błąd popełniany z powodu dyskretyzacji miał inny niż losowy charakter i dlatego jakikolwiek konkretny wynik sam w sobie jest mało użyteczny: wszak rozdzielczość użytej przez nas siatki nigdy nie będzie wystarczająca, by uchwycić wszelkie wahnięcia  $f$ ! Jednak zadanie będzie dla nas dobrym pretekstem do tego, by

- pokazać wygładzający efekt dyfuzji
- zobaczyć ograniczenia bezpośredniego *solvera* układów równań z macierzami rzadkimi na *jakimkolwiek* konkretnym przykładzie.

```
X = 1; Y = 2; Z = 3; % fizyczne rozmiary kostki
Nx = 30; Ny = 30; Nz = 30;

hx = X/(Nx+1);
hy = Y/(Ny+1);
hz = Z/(Nz+1);

px = hx*[0:Nx+1]; py = hy*[0:Ny+1]; pz = hz*[0:Nz+1]; % prawdziwe współrzędne węzłów (łącznie z brzegiem obszaru)

% macierze pomocnicze dla zadań jednowymiarowych
Tx = lap1ddset(Nx,hx); Ty = lap1ddset(Ny,hy); Tz = lap1ddset(Nz,hz);
lx = speye(size(Tx)); ly = speye(size(Ty)); lz = speye(size(Tz));

% macierz laplasjanu
S = kron(lz, kron(ly, Tx)) + kron(lz, kron(Ty, lx)) + kron(Tz, kron(ly, lx));

% prawa strona i jej wizualizacja
F = 1 + 0.1*(2*rand(Nx+2,Ny+2,Nz+2)-1);
slice(px, py, pz, permute(F, [2 1 3]), X/2,Y/2, Z/2);
xlabel('x'); ylabel('y'); zlabel('z'); grid on; title('Prawa_strona');
pause(1);

F = F(2:Nx+1,2:Ny+1,2:Nz+1); % ograniczamy się do węzłów wewnętrznych

% wyznaczamy rozwiązanie
U = zeros(Nx+2,Ny+2,Nz+2);
U(2:Nx+1,2:Ny+1,2:Nz+1) = reshape(S \ F(:),Nx,Ny,Nz);

% wizualizacja rozwiązania
slice(px, py, pz, permute(U, [2 1 3]), X/2,Y/2, Z/2);
xlabel('x'); ylabel('y'); zlabel('z'); grid on; title('Rozwiazanie');
pause(1);
```

Zwróćmy uwagę na to, jak grubą siatkę wzięliśmy w naszym przykładowym kodzie. Gdyby bowiem zagęścić ją dwukrotnie, czas potrzebny na uzyskanie rozwiązania znacząco wydłuży się; próba wyznaczenia rozwiązania na siatce  $80 \times 80 \times 80$  kończy się<sup>5</sup> komunikatem o błędzie z powodu braku wystarczających zasobów.

### Krytyka standardowego solvera bezpośredniego

Jakkolwiek stosowany przez nas *solver* bezpośredni całkiem dobrze sobie radzi z zadaniami dwuwymiarowymi, w których liczba węzłów sięga nawet  $500 \times 500$ , to już dla zadań rozmiaru

<sup>5</sup> Na komputerze z 2GB pamięci RAM. Na szybkiej maszynie z „nieograniczonymi” zasobami (wieloprocesorowy Intel Xeon 2.4GHz, 64GB RAM), rozwiązanie udaje się policzyć, owszem, wykorzystując w tym celu (w MATLABie) ponad 8.5 GB pamięci i 400 sekund czasu rzeczywistego.

1000 × 1000 staje się mało efektywny, a jego wymagania pamięciowe stają się duże. Sytuacja staje się jeszcze bardziej dramatyczna, gdy przejdziemy do zadań trójwymiarowych, co mieliśmy okazję oglądać w ostatnim przykładzie. Jasne jest, że stosowane przez bezpośredni *solver* algorytmy *reorderingu* niewiadomych i równań są nieskuteczne w przypadku dyskretyzacji trójwymiarowych — nawet tak regularnych jak nasza.

### Solver oparty na FFT

W przypadku tak specyficznej macierzy, jaką jest macierz laplasjanu, można na szczęście wskazać specjalne metody jej rozwiązywania. To typowa i charakterystyczna dla obliczeń naukowych sytuacja, kiedy do konkretnego zadania można dobrać niestandardową metodę, która będzie znacznie skuteczniejsza od metod ogólnego przeznaczenia. Często taka metoda będzie motywowana fizycznymi cechami zadania; w naszym przypadku lepsza metoda będzie wynikać wyłącznie z matematycznych własności zadania.

Otóż okazuje się, że znane są jawne wzory na czynniki rozkładu macierzy  $T_N$  na wartości i wektory własne,

$$T_N = Q_N \Lambda_N Q_N^T,$$

skąd wynika, że

— w przypadku dwuwymiarowym:

$$P_N = Q_{N_y} Q_{N_x} \underbrace{(I_{N_y} \otimes \Lambda_{N_x} + \Lambda_{N_y} \otimes I_{N_x})}_{\text{macierz diagonalna}} Q_{N_x}^T Q_{N_y}^T.$$

— w przypadku trójwymiarowym:

$$S_N = Q_{N_z} Q_{N_y} Q_{N_x} \underbrace{(I_{N_z} \otimes I_{N_y} \otimes \Lambda_{N_x} + I_{N_z} \otimes \Lambda_{N_y} \otimes I_{N_x} + \Lambda_{N_z} \otimes I_{N_y} \otimes I_{N_x})}_{\text{macierz diagonalna}} Q_{N_x}^T Q_{N_y}^T Q_{N_z}^T.$$

Rozwiązanie układu z macierzą diagonalną kosztuje tylko  $N$  operacji, natomiast mnożenia przez macierze  $Q_N$  można łatwo zrealizować (niskim kosztem) za pomocą algorytmu FFT, dostępnego w Octave. Więcej informacji na ten temat (w tym — informację o *jeszcze szybszym* algorytmie!) można znaleźć w wykładach z [Matematyki Obliczeniowej II](#).

**Przykład 8.1** (Trójwymiarowe równanie Poissona przez FFT). Pokażemy, jak rozwiązywać trójwymiarowe równanie Poissona (por. ćwiczenie 8.6) z wykorzystaniem *solwera* opartego na FFT. Jest to niewielka modyfikacja kodu z ćwiczenia 8.6: niepotrzebne fragmenty zostały wykomentowane, a nowo wstawiane części oznakowano komentarzem NOWE.

```
X = 1; Y = 2; Z = 3; % fizyczne rozmiary kostki
Nx = 100; Ny = 100; Nz = 100; scale = 0.1;

hx = X/(Nx+1);
hy = Y/(Ny+1);
hz = Z/(Nz+1);

px = hx*[0:Nx+1]; py = hy*[0:Ny+1]; pz = hz*[0:Nz+1]; % prawdziwe współrzędne węzłów (łącznie z brzegiem obszaru)

% macierze pomocnicze dla zadan jednowymiarowych
% Tx = lap1ddset(Nx,hx); Ty = lap1ddset(Ny,hy); Tz = lap1ddset(Nz,hz); % potrzebne tylko dla metody bezpośredniej
lx = speye(Nx,Nx); ly = speye(Ny,Ny); lz = speye(Nz,Nz);

% NOWE!
Lx = spdiags(fftsetd(Nx,hx),0,Nx,Nx); Ly = spdiags(fftsetd(Ny,hy),0,Ny,Ny); Lz = spdiags(fftsetd(Nz,hz),0,Nz,Nz); % macierze Laplasjana
```

```

% macierz laplasjanu, tylko dla metody bezposredniej
% S = kron(Iz, kron(Iy, Tx)) + kron(Iz, kron(Ty, Ix)) + kron(Tz, kron(Iy, Ix));

% NOWE!
% macierz diagonalna zadania 3D, tylko dla metody FFT
global Lambda; % gloablna, bo wykorzystywana przez fftsolve3dd()
Lambda = kron(Iz, kron(Iy, Lx)) + kron(Iz, kron(Ly, Ix)) + kron(Lz, kron(Iy, Ix));

% prawa strona
F = 1 + scale*(2*rand(Nx+2,Ny+2,Nz+2)-1);
slice(px, py, pz, permute(F, [2 1 3]), X/2, Y/2, Z/2);
xlabel('x'); ylabel('y'); zlabel('z'); grid on; title('Prawa_strona');
pause(1);

F = F(2:Nx+1,2:Ny+1,2:Nz+1);

% tylko dla metody bezposredniej
% U = zeros(Nx+2,Ny+2,Nz+2);
% U(2:Nx+1,2:Ny+1,2:Nz+1) = reshape(S \ F(:),Nx,Ny,Nz); % rozwiazanie

% NOWE!
% macierz diagonalna zadania 3D, tylko dla metody FFT
U = zeros(Nx+2,Ny+2,Nz+2);
tic; U(2:Nx+1,2:Ny+1,2:Nz+1) = fftsolve3dd(F); toc % rozwiazanie FFT

% wizualizacja
slice(px, py, pz, permute(U, [2 1 3]), X/2, Y/2, Z/2);
xlabel('x'); ylabel('y'); zlabel('z'); grid on; title('Rozwiazanie');
pause(1);

```

Powyższy skrypt korzysta z pewnych dodatkowych funkcji: `fftsolve3dd` i `fftsetd`; poniżej ich listingi.

```

function V = fftsolve3dd(F)
% Rozwiazuje, korzystajac z FFT, uklad rownan liniowych Lap*V = F,
% gdzie V,F sa macierzami wartosci na siatce kwadratowej.
% Warunki brzegowe Dirichleta: jednorodne.
global Lambda; % wartosci wlasne

[Nx, Ny, Nz] = size(F);

F = fftmult3d(F); %F = ZX*F*ZY;

V = Lambda\F(:);

V = reshape(V, Nx, Ny, Nz);

V = fftmult3d(V); %V = ZX*V*ZY;

end

```

```

function Lambda = fftsetd(N,h)
%
% Wartosci wlasne dla 1-d Laplasjanu z war brzeg. Dirichleta

```

```
Lambda = (4/h^2)*(sin(0.5*pi*[1:N]/(N+1))).^2; % exact eigenvalues
end
```

Nagle... realne staje się szybkie wyznaczenie rozwiązania zadania trójwymiarowego, nie tylko na siatce  $100 \times 100 \times 100$  (to milion niewiadomych!) w czasie poniżej 1 sekundy (sic!) na komputerze Core 2 Duo 2.4 GHz z pamięcią 2GB. Nawet zadanie postawione na siatce  $200 \times 200 \times 200$  wciąż liczy się w więcej niż przyzwoitym czasie: poniżej 7 sekund!

Kluczowa dla szybkości działania *solvera* jest funkcja `fftsolve3dd`, wykorzystująca umiejętność Octave przykładania FFT „wzdłuż” zadanego wymiaru macierzy.

**Ćwiczenie 8.7.** Zmodyfikuj powyższy kod oraz funkcję `fftsolve3dd` tak, by mogła działać w przypadku dyskretyzacji dwuwymiarowego równania Poissona.

**Ćwiczenia testowe 8.8.** 1. Czy warto zastosować powyższy algorytm do rozwiązywania jednowymiarowego zadania Poissona?

**NIE.** Komentarz do odpowiedzi prawidłowej: *Metoda oparta na eliminacji Gaussa, wykorzystująca strukturę macierzy trójdzielnej, jest w tym wypadku optymalna.* Komentarz do odpowiedzi błędnej: *Pomyśl, ile kosztuje rozwiązanie układu równań z macierzą trójdzielną.*

2. Czy w przypadku dwuwymiarowym opisana powyżej metoda oparta na FFT będzie dawała rozwiązanie obciążone większym błędem niż w sytuacji gdy zastosujemy w Octave standardowy operator rozwiązywania równań liniowych?

**NIE.** Komentarz do odpowiedzi prawidłowej: *Odpowiedź, przynajmniej teoretycznie, brzmi: nie. Obie metody rozwiązują ten sam układ równań i obie są metodami bezpośrednimi: w arytmetyce dokładnej dają dokładne rozwiązanie. Ponieważ w praktyce obliczenia wykonujemy w arytmetyce podwójnej precyzji, a każda z metod wykonuje inne rachunki, należy spodziewać się niewielkiej różnicy, „na poziomie precyzji arytmetyki”. Większy wpływ na wynik może mieć złe uwarunkowanie samej macierzy, a nie różnice między solverami. Patrz także [12].* Komentarz do odpowiedzi błędnej: *To jeszcze nie uzasadnienie, ale... która z metod wyznaczy rozwiązanie niższym kosztem. Zajrzyj do [12].*

**Ćwiczenie 8.9.** Sprawdź, czy metoda iteracyjna — taka jak na przykład PCG, o której więcej dowiesz się z wykładu z [Matematyki Obliczeniowej II](#), a która jest implementowana w Octave komendą `pcg` — jest rozsądną alternatywą dla *solvera* FFT.

*Wskazówka.* W przypadku zadania trójwymiarowego należy użyć komendy

```
U = zeros(Nx+2,Ny+2,Nz+2);
[Uin, flag, relres, iter, resvec, eigest] = ...
    pcg(S,F(:), (1e-3)*min([hx,hy,hz])^2, 50);
U(2:Nx+1,2:Ny+1,2:Nz+1) = reshape(Uin,Nx,Ny,Nz);
disp(['Uwarunkowanie_S= ', num2str(eigest(2)/eigest(1))]);
```

Jakie jest uwarunkowanie tej macierzy? (Odpowiedź na to pytanie znajdziesz na przykład znów w wykładzie z [Matematyki Obliczeniowej II](#).) Co to oznacza dla *solvera* iteracyjnego, takiego jak PCG?

## 8.2. Równanie dyfuzji

Wykorzystywany przez nas *solver* FFT ma niestety poważne ograniczenia, które uniemożliwiają zastosowanie go wprost do zadania (8.1), w którym mamy co prawda do czynienia z jednorodną siatką na kostce  $\Omega$ , ale jednocześnie współczynnik dyfuzji nie jest stały w całym obszarze: aby go zastosować, musieliśmy założyć, że mamy do czynienia z równaniem Poissona, bo tylko dla niego znaleźliśmy wartości i wektory własne odpowiednich macierzy.

Okazuje się wszakże, iż możemy efektywnie skorzystać z FFT nawet w przypadku zmieniającego się współczynnika dyfuzji. Jednak wtedy musimy podejść do zadania z *jeszcze innej* strony: używając metody iteracyjnej PCG do rozwiązywania zadania i wykorzystując jednocześnie bezpośredni *solver* oparty na FFT w charakterze wydajnego *operatora ściskającego*<sup>6</sup>, dzięki czemu metoda PCG będzie szybko zbieżna!

Zwróćmy uwagę, że wiele metod iteracyjnych — w tym metoda PCG — rozwiązywania układu równań liniowych

$$Ax = b$$

nie wymaga wcale *konstrukcji* macierzy  $A$ ; w zupełności wystarczy im funkcja, wyznaczająca dla zadanego wektora  $v$  iloczyn  $Av$  (takie metody iteracyjne nazywamy metodami operatorowymi). W naszym przypadku będzie to wielkim ułatwieniem, gdyż obliczenie działania dyskretnego operatora dyfuzji na zadanej funkcji siatkowej będzie łatwe do osiągnięcia — w przeciwieństwie do jawnej konstrukcji macierzy takiego operatora. To jeszcze jeden powód, by zastosować metodę PCG.

Aby przetestować nasze nowe podejście, zaczniemy od przypadku jednowymiarowego, uwzględniając (wreszcie!) przypadek, gdy warunek brzegowy *nie jest* jednorodny.

### 8.2.1. Dyskretyzacja zadania jednowymiarowego

Gdy  $\Omega$  jest odcinkiem  $(0, X)$ , (8.1) staje się równaniem różniczkowym zwyczajnym z warunkiem brzegowym:

$$-\frac{d}{dx}(a(x)\frac{d}{dx}u(x)) = f(x) \quad \forall x \in (0, X), \quad (8.18)$$

$$u(0) = g(0), \quad u(X) = g(X). \quad (8.19)$$

Aby znaleźć jego przybliżone rozwiązanie, możemy na przykład zastąpić równanie różniczkowe odpowiadającym mu równaniem *różnicowym*. Okazuje się, że prostoduszne

$$-\frac{a_{i-1}u_{i-1} - 2a_i u_i + a_{i+1}u_{i+1}}{h_x^2} = f_i$$

dość kiepsko aproksymuje nasze równanie, a znacznie lepsze własności (*wiedza nie szkodzi!*) ma aproksymacja postaci

$$-\frac{a_{i-1/2}u_{i-1} - (a_{i-1/2} + a_{i+1/2})u_i + a_{i+1/2}u_{i+1}}{h_x^2} = f_i \quad \forall i = 1, \dots, N_x, \quad (8.20)$$

$$u_0 = g_0, \quad u_{N_x+1} = g_{N_x+1}, \quad (8.21)$$

gdzie za  $a_{i+1/2}$  można wziąć na przykład

$$a_{i+1/2} = \frac{a_i + a_{i+1}}{2} \quad \text{lub} \quad a_{i+1/2} = a(x_{i+1/2}).$$

gdzie, zgodnie z intuicją,  $x_{i+1/2} = \frac{x_i + x_{i+1}}{2}$ . Widzimy więc, że obliczenie działania dyskretnego operatora dyfuzji  $D_{N_x} : \mathbb{R}^{N_x+2} \rightarrow \mathbb{R}^{N_x}$  na wektorze  $U_{N_x}$  (dane wzorem (8.20)), wymaga znajomości wartości także  $U_{N_x}$  na brzegu obszaru i prowadzi do wyznaczenia wartości wyniku jedynie w węzłach wewnętrznych.

<sup>6</sup> Więcej szczegółów na temat PCG, ściskania i macierzy ściskających można poznać na wykładzie z [Matematyki Obliczeniowej II](#).



### Uwzględnienie warunków brzegowych

W naszym zadaniu poszukujemy rozwiązania postaci

$$U_{N_x} = (g_0, \underbrace{u_1, \dots, u_{N_x}}_{\text{realne niewiadome}}, g_{N_x+1})^T$$

takiego, że

$$D_{N_x} U_{N_x} = (f_1, \dots, f_{N_x})^T.$$

Powyższe zadanie, z niejednorodnym warunkiem brzegowym, bardzo łatwo sprowadzić do zadania z warunkiem jednorodnym. Rzeczywiście, oznaczając

$$\tilde{U}_{N_x} = U_{N_x} - (g_0, \underbrace{0, \dots, 0}_{\text{wewnętrzne}}, g_{N_x+1})^T$$

mamy, że  $\tilde{U}_{N_x}$  na brzegu przyjmuje wartości równe zero oraz spełnia

$$D_{N_x} \tilde{U}_{N_x} = (f_1, \dots, f_{N_x})^T - D_{N_x} (g_0, 0, \dots, 0, g_{N_x+1})^T. \quad (8.22)$$

W ten sposób wystarczy zmodyfikować prawą stronę równania, by potem przez rozwiązanie równania z jednorodnym warunkiem brzegowym wyznaczyć wartości rozwiązania we wszystkich punktach wewnętrznych siatki.

Oto więc, jak można byłoby zaimplementować działanie operatora  $D_{N_x}$  w Octave:

```
function F = multdiff1dd(Uin, h, A, U)
%
% wyznacza F = -(a(x) u'(x))', dla dyskretyzacji
%
% Uin - wartości u(x) we wnętrzu siatki
% h - długość kroku siatki
% A - wartości a(x) we wszystkich węzłach pośrednich siatki
% domyślnie a(x) = 1
% U - wartości zerowego rozszerzenia warunku brzegowego na wszystkie węzły siatki
% domyślnie U(x) = 0

if nargin < 4 % domyślnie jednorodny warunek brzegowy
    U = zeros(length(Uin)+2,1);
end

[Nx, Ny] = size(U);

if nargin < 3 % domyślnie a(x) = 1
    Ax = ones(Nx-1,1);
else
    Ax = A;
end

ix = 2:Nx-1; % indeksy odpowiadające wartościom z wnętrza

U(2:Nx-1) = Uin;
F = (-U(ix-1).*Ax(ix-1) ...
    + U(ix).*(Ax(ix-1)+Ax(ix)) ...
    - U(ix+1).*Ax(ix))/(h(1)^2);
end
```

Jeśli więc chcemy wyznaczyć wynik działania  $D_{N_x}$  na

- tablicy  $U$  o wartościach zadanych *wyłącznie* w *wewnętrznych* węzłach siatki i domyślnej wartości zero na brzegu, użylibyśmy wywołania

```
F = multdiff1dd(U, hx, Ax);
```

- tablicy o wartościach w *wewnętrznych* węzłach siatki równej zero i wartościach na brzegu określonych przez tablicę  $G$  (rozmiaru  $N_x + 2$ ), użylibyśmy wywołania

```
F = multdiff1dd(zeros(Nx,1), hx, Ax, G);
```

Ostatnie wywołanie pasuje jak ulał do wyznaczenia poprawki na prawą stronę  $D_{N_x}(g_0, 0, \dots, 0, g_{N+1})^T$ , por. (8.22).

**Ćwiczenie 8.10.** Pamiętając o konieczności nieustannego weryfikowania poprawności zaimplementowanych funkcji, przygotuj i przeprowadź serię testów sprawdzających funkcję `multdiff1dd`.

*Wskazówka.* Warto wykorzystać w tym celu m.in. napisane wcześniej funkcje testujące/rozwiązujące równanie Poissona z jednorodnym warunkiem brzegowym.

## 8.2.2. Rozwiązanie równania dyfuzji w obszarze jednowymiarowym

Zgodnie z wcześniej przyjętą strategią, zastosujemy iteracyjną metodę rozwiązywania równania. Dla ustalenia uwagi przyjmijmy, że  $a(x) = x + 1$  oraz  $f(x) = g(x) = 1$ .

```
A = pX+1; % definicja wartości współczynnika dyfuzji w węzłach siatki
Ax = 0.5*(A(1:end-1) + A(2:end));
multdiff = @(x) multdiff1dd(x, hx, Ax); % funkcja przykładowa operator dyfuzji z zerowym warunkiem brzegowym
precond = @(x) Tx\ x; % jako operator ściskający wybieramy rozwiązanie r- nia Poissona

F = ones(Nx,1);
U = ones(Nx+2,1);
F = F - multdiff1dd(zeros(Nx,1), hx, Ax, U); % uwzględnienie warunku brzegowego
[Uin, flag, relres, iter, resvec, eigest] = ...
    pcg(multdiff, F(:), (1e-3)*hx^2, 50, precond);
U(2:Nx+1) = Uin;
```

**Ćwiczenie 8.11.** Czy w przypadku jednowymiarowej dyfuzji nie można byłoby w prosty sposób użyć *solvera* bezpośredniego?

*Rozwiązanie.* Faktycznie, mamy

$$D_{N_x} = \frac{1}{h_x^2} \begin{pmatrix} -a_{1/2} & a_{1/2} + a_{3/2} & -a_{3/2} & & & \\ & -a_{3/2} & a_{3/2} + a_{5/2} & -a_{5/2} & & \\ & & \ddots & \ddots & \ddots & \\ & & & -a_{(N_x-2)/2} & a_{(N_x-2)/2} + a_{(N_x-1)/2} & -a_{(N_x-1)/2} \\ & & & & -a_{(N_x-1)/2} & a_{(N_x-1)/2} + a_{(N_x+1)/2} & -a_{(N_x+1)/2} \end{pmatrix}$$

A więc można (i nawet: trzeba!) w tym wypadku użyć *solvera* bezpośredniego. My implementujemy *solver* iteracyjny jedynie w celu oswojenia i sprawdzenia podejścia związanego z wykorzystaniem metody iteracyjnej...

**Ćwiczenie 8.12.** Pamiętając o konieczności nieustannego weryfikowania poprawności zaimplementowanych funkcji, przygotuj i przeprowadź serię testów sprawdzających opracowaną metodę.

*Wskazówka.* Przede wszystkim, warto sprawdzić, co dzieje się w przypadku  $a(x) \equiv 1$ , czyli w znanym nam doskonale przypadku równania Poissona. Czy dla testowanych prawych stron dostajemy zawsze niemal identyczne rozwiązania? (doskonałą prawą stroną jest wektor losowy!) Czy zawsze dostajemy zbieżność PCG w jednej iteracji? Co dzieje się, gdy zwiększamy  $N_x$ ? Następnie należy koniecznie przeprowadzić serię testów dla  $a(x) \neq 1$  (pamiętając o tym by wybierane przez nas  $a$  było ostro odgraniczone od zera) w sposób analogiczny jak to robiliśmy w przypadku dyskretnego operatora Laplace'a. Czy gdy  $\frac{\max |a(x)|}{\min |a(x)|} \gg 1$ , to zbieżność pogarsza się? (Powinna!)

**Ćwiczenie 8.13.** Funkcja mnożąca wektor przez operator ściskający powinna być możliwie tania, gdyż będziemy ją wywoływać wiele razy. Nasza funkcja:

```
precond = @(x) Tx \ x;
```

ma tę wadę, że za każdym wywołaniem dokonuje rozkładu LU macierzy  $T_x$ . Pomyśl, jak to usprawnić.

*Rozwiązanie.* Wyznaczając na etapie *preprocessingu* rozkład LU (lub Cholesky'ego)

```
Rx = spchol(Tx); % rozkład Cholesky'ego: Rx'*Rx = Tx
precond = @(x) Rx \ (Rx' \ x);
```

### 8.2.3. Dyskretyzacja dwuwymiarowego operatora dyfuzji

Gdybyśmy od razu chcieli atakować zadanie (8.1) w przypadku dwu- lub więcejwymiarowym, mogłoby nam być dosyć trudno. Teraz jednak, po przepracowaniu przypadku jednowymiarowego, sprawa powinna potoczyć się wartko. Będziemy musieli pamiętać o dwoistości spojrzenia na niewiadome: raz będziemy je traktować jako (dwuwymiarową!) tablicę wartości w węzłach (tak będzie nam wygodniej zapisać działanie operatora dyfuzji), innym razem będziemy musieli rozwinąć tablicę w jeden długi wektor (tak działa funkcja `pcg` w Octave).

Przez analogię do przypadku jednowymiarowego, nasza dyskretyzacja będzie miała postać:

$$\begin{aligned}
 & - \frac{a_{i-1/2,j}u_{i-1,j} - (a_{i-1/2,j} + a_{i+1/2,j})u_{i,j} + a_{i+1/2,j}u_{i+1,j}}{h_x^2} \\
 & - \frac{a_{i,j-1/2}u_{i,j-1} - (a_{i,j-1/2} + a_{i,j+1/2})u_{i,j} + a_{i,j+1/2}u_{i,j+1}}{h_y^2} \\
 & = f_{i,j} \quad \forall i = 1, \dots, N_x, j = 1, \dots, N_y,
 \end{aligned}$$

z warunkiem brzegowym

$$u_{0,j} = g_{0,j}, \quad u_{N_x+1,j} = g_{N_x+1,j}, \quad u_{i,0} = g_{i,0}, \quad u_{i,N_y+1} = g_{i,N_y+1}, \quad \forall i = 1, \dots, N_x, j = 1, \dots, N_y.$$

Mamy do wyznaczenia  $N = N_x \cdot N_y$  niewiadomych, które w naturalny sposób układają się w macierz

$$U_N = \begin{pmatrix} u_{1,1} & u_{1,2} & u_{1,3} & \dots & u_{1,N_y} \\ u_{2,1} & u_{2,2} & u_{2,3} & \dots & u_{2,N_y} \\ \vdots & & & & \\ u_{N_x,1} & u_{N_x,2} & u_{N_x,3} & \dots & u_{N_x,N_y} \end{pmatrix}.$$

Zauważmy, że aby na przykład wyznaczyć iloraz różnicowy

$$F = \frac{-u_{i-1,j} + 2u_{i,j} - u_{i+1,j}}{h_x^2},$$

nie musimy używać żadnej macierzy; możemy po prostu wykonać działanie, które, nieco nadużywając notacji Octave<sup>7</sup>, zapiszemy

```
F = -U( 0:Nx-1, :) + 2*U( 1:Nx, :) - U( 2:Nx+1, :);
F = F/hx^2;
```

Analogicznie, dysponując tablicą  $A_x$  zawierającą wartości  $a_{i+1/2,j}$  dla  $i = 0, \dots, N_x$  oraz  $j = 1, \dots, N_y$ , iloraz różnicowy

$$F = \frac{-a_{i-1/2,j}u_{i-1,j} + (a_{i-1/2,j} + a_{i+1/2,j})u_{i,j} - a_{i+1/2,j}u_{i+1,j}}{h_x^2}$$

wyznamy korzystając z operatorów tablicowego mnożenia,

```
F = -Ax(0:Nx-1).*U( 0:Nx-1, :) + (Ax(0:Nx-1)+Ax(1:Nx)).*U( 1:Nx, :) - Ax(1:Nx).*U( 2:Nx+1, :);
F = F/hx^2;
```

Powyższa obserwacja pozwoli nam efektywnie — bez użycia pętli i instrukcji warunkowych — obliczyć *wynik* działania dyskretnego dwuwymiarowego operatora dyfuzji na tablicy wartości funkcji siatkowej w węzłach.

**Ćwiczenie 8.14.** Zaimplementuj funkcję  $F = \text{multidiff2dd}(U_{in}, h, A_x, A_y, U)$ , będącą analogonem opracowanej wcześniej funkcji jednowymiarowej. Niech  $a(x, y) = xy + 1$ . Wyznacz i zobrazuj wynik działania dyskretyzacji operatora dyfuzji  $-\text{div}(a(x, y)\nabla u(x, y))$  w przypadku, gdy  $u(x, y) = xy$  na  $\bar{\Omega} = [0, 2] \times [0, 3]$ .

*Rozwiązanie.* Najpierw implementacja operatora dyfuzji:

```
function F = multidiff2dd(Uin, h, A, U)
%
% wyznacza F = -div ( a(x,y) grad u(x,y) ), dla dyskretyzacji
%
% Uin - wartości u(x,y) we wnętrzu siatki
% h - długość kroku siatki: h = [hx,hy]
% A - wartości a(x,y) we wszystkich punktach pośrednich siatki: A = {Ax, Ay}
% domyślnie a(x) = 1
% U - wartości zerowego rozszerzenia warunku brzegowego na wszystkie węzły siatki
% domyślnie U(x) = 0

if nargin < 4 % domyślnie jednorodny warunek brzegowy
    U = zeros(size(Uin)+2);
end

[Nx, Ny] = size(U);

if nargin < 3 % domyślnie a(x) = 1
    Ax = ones(Nx-1,Ny-2);
    Ay = ones(Nx-2,Ny-1);
else
    Ax = A{1};
    Ay = A{2};
end

ix = 2:Nx-1; iy = 2:Ny-1; % indeksy odpowiadające wartościom z wnętrza
U(2:Nx-1,2:Ny-1) = Uin;
```

<sup>7</sup> W Octave nie wolno indeksować macierzy od zera.

```
F = (-U(ix-1,iy).*Ax(ix-1,:) + U(ix,iy).*(Ax(ix-1,:)+Ax(ix,:)) - U(ix+1,iy).*Ax(ix,:))/(h(1)^2) + ...
    (-U(ix,iy-1).*Ay(:,iy-1) + U(ix,iy).*(Ay(:,iy-1)+Ay(:,iy)) - U(ix,iy+1).*Ay(:,iy))/(h(2)^2);
end
```

Podczas pisania tej funkcji musimy bardzo uważać, by uniknąć pomyłek przy przesunięciach indeksów. Zwróćmy także uwagę na pewną rozrzutność w gospodarowaniu pamięcią: wektory pomocnicze  $A_x$  i  $A_y$  są (formalnie niepotrzebnymi) kopiami pól struktury  $A$ : użyliśmy ich dla większej czytelności kodu.

Dalej, po podstawowych testach (które tutaj pominiemy wierząc, że Czytelnik samodzielnie i rutynowo je przeprowadzi), wykorzystamy świeżo napisaną funkcję w konkretnym przypadku opisanym w zadaniu.

```
X = 2; Y = 3;
Nx = 151; Ny = 153;
hx = X/(Nx+1); hy = Y/(Ny+1);

px = hx*[0:Nx+1]; py = hy*[0:Ny+1];

[pX, pY] = meshgrid(px,py);
A = pX.*pY+1;
contour(px, py, A ); title('A(x,y)'); pause(1);
A = permute(A, [2 1]); % przechodzimy na nasz układ tablicowy
Ax = 0.5*(A(1:end-1,2:end-1) + A(2:end,2:end-1));
Ay = 0.5*(A(2:end-1,1:end-1) + A(2:end-1,2:end));

U = permute(pX.*pY, [2 1]);
F = multdiff2dd(U(2:Nx+1,2:Ny+1),[hx,hy],{Ax,Ay},U);
contour(px(2:Nx+1), py(2:Ny+1), permute(F, [2 1]) ); title('-div_⊥(a(x,y)_grad_⊥u(x,y))'); pause(1);
```

#### 8.2.4. Rozwiązanie równania dyfuzji w obszarze dwuwymiarowym

Sama procedura rozwiązywania równania nie zmieni się zanadto w stosunku do przypadku jednowymiarowego; dla ustalenia uwagi przyjmiemy, że  $a(x, y) = xy + 1$  oraz  $f(x, y) = g(x, y) = 1$ , a wszystkie pozostałe parametry zadania są takie same, jak w uprzednio rozważanym zagadnieniu Poissona w prostokącie.

```
%
% definicje siatki oraz macierzy Lx, Ly, lx, ly jak w przypadku r-nia Poissona –
% – tutaj pominięte!
%

global Lambda;
Lambda = kron(Ly,Lx) + kron(Ly,lx);

% definicja wartości współczynnika dyfuzji w węzłach siatki
[pX, pY] = meshgrid(px,py);
A = pX.*pY+1;
A = permute(A, [2 1]);
Ax = 0.5*(A(1:end-1,2:end-1) + A(2:end,2:end-1));
Ay = 0.5*(A(2:end-1,1:end-1) + A(2:end-1,2:end));
% definicja operatorów niezbędnych do rozwiązania zadania
multdiff = @(x) reshape(multdiff2dd(reshape(x,Nx,Ny), [hx,hy], {Ax,Ay}), Nx*Ny, 1); % funkcja przykładowa operator dyfuzji
precond = @(x) reshape(fftsolve2dd(reshape(x, Nx, Ny)), Nx*Ny, 1); % jako operator ściskający wybieramy rozwiązanie r-nia Poissona
```

```

F = ones(Nx,Ny);
U = ones(Nx+2,Ny+2);
F = F - multdiff2dd(zeros(size(F)), [hx,hy], {Ax,Ay}, U); % uwzględnienie warunku brzegowego
[Uin, flag, relres, iter, resvec, eigest] = ...
    pcg(multdiff, F(:), (1e-3)*min([hx,hy])^2, 50, precondition);
U(2:Nx+1,2:Ny+1) = reshape(Uin,Nx,Ny);

contour(px, py, permute(U, [2 1]) );
xlabel('x'); ylabel('y');

```

Brakującą funkcję `fftsolve2dd` bez trudu opracujesz, mając za wzór funkcję `fftsolve3dd` z przykładu 8.1.

**Ćwiczenie 8.15** (trudne). Zaimplementuj w Octave funkcję rozwiązującą trójwymiarowe zadanie dyfuzji.

**Ćwiczenie 8.16** (trudne). Zaimplementuj w Octave funkcję rozwiązującą na kostce  $d$ -wymiarowe ( $d \in \{1, 2, 3\}$ ) zadanie dyfuzji z warunkiem brzegowym *Neumanna*.

### 8.3. Uwagi i uzupełnienia

W przypadku zadań trudniejszych, na przykład obszarów o bardziej skomplikowanej geometrii, warto skorzystać z bardziej wyspecjalizowanych narzędzi. Jedną z możliwości jest użycie biblioteki [Deal.II](#) napisanej w języku C++ i pozwalającej dyskretyzować równania różniczkowe różnych typów przy użyciu metody elementu skończonego. Zawiera on w sobie także kilka narzędzi do rozwiązywania zadań dyskretnych; ma także możliwość współpracy z innymi pakietami pomocniczymi, takimi jak zestaw procedur rozwiązywania zadań dyskretnych na komputerach wieloprocessorowych [PETSc](#). Wadą tego podejścia jest konieczność nauczania się i zrozumienia tej skomplikowanej biblioteki, co jest bardzo czasochłonne.

Tej wady nie mają proste narzędzia, takie jak program [freeFEM++](#), ale ich stosowalność jest mocno ograniczona.

**Ćwiczenie 8.17** (trudne). Zaimplementuj w C/C++ funkcję rozwiązującą dwu- lub trójwymiarowe zadanie dyfuzji ze zmiennym współczynnikiem, z wykorzystaniem biblioteki [Deal.II](#).

**Ćwiczenie 8.18** (trudne). Zaimplementuj w Octave lub w C++ funkcję rozwiązującą anizotropowe zadanie dyfuzji w  $\Omega = (0, 1)^d$  ze zmiennym współczynnikiem,

$$-\operatorname{div}(A(x)\nabla u(x)) = f(x), \quad x \in \Omega,$$

z warunkiem brzegowym Dirichleta. Tutaj, dla zadanego  $x \in \Omega$ , macierz  $A(x)$  jest dodatnio określoną macierzą rozmiaru  $d \times d$ . Rozważ  $d \in \{1, 2, 3\}$ .

## 9. Równanie logistyczne z opóźnieniem

Klasyczne modele oparte na równaniach różniczkowych zwyczajnych zakładają, że w danej chwili reakcja systemu (zmiana wartości) jest natychmiastowa i zależy od jej wartości w tym momencie (czyli, że  $y'(t) = f(t, y(t))$ ). Jednak bywają sytuacje, kiedy takie założenie jest nierealistyczne i należy uwzględnić fakt, iż — wskutek swego rodzaju bezwładności układu — wartość pochodnej rozwiązania może zależeć nie tylko od jego bieżącej kondycji, lecz również od wartości z chwil *wcześniejszych*. W pierwszym przybliżeniu taka koncepcja prowadzi to do tzw. równań różniczkowych ze stałym opóźnieniem  $\tau \geq 0$ :

$$y'(t) = f(t, y(t), y(t - \tau)), \quad t \in (t_{\min} t_{\max}].$$

Pakiet OdePkg, rozszerzający możliwości Octave i dostępny w Octave-Forge [2], zawiera w sobie kilka *solverów* równań różniczkowych ze stałym opóźnieniem  $\tau$ . Drobną ich wadą jest to, że dopuszczają jedynie stały warunek początkowy (w równaniach z opóźnieniem warunek początkowy musi być zadany nie w jednym punkcie, ale na odcinku,  $[t_{\min} - \tau, t_{\min}]$ ). Wszystkie *solvery* wykorzystują tzw. włożone jawne metody Rungego–Kutty, od metody niskiego rzędu ode23d, po metodę wysokiego rzędu ode78d. Warto pamiętać, że w przypadku równań z opóźnieniem rozwiązania mogą nie być zbyt gładkie i wówczas korzystniej może być stosować schemat *niższego* rzędu.

Sposób wykorzystania pakietu jest bardzo dokładnie opisany w [25], na którym, jak się wydaje, opiera się zestaw *solverów* z OdePkg. My ograniczymy się do krótkiego omówienia sposobu rozwiązania jednego z najprostszych równań.

### 9.1. Sformułowanie zadania i implementacja

Rozważmy równanie postaci

$$y'(t) = (a - y(t - 1)) \cdot y(t), \quad \text{dla } t \in (0, T], \quad (9.1)$$

$$y(t) = 0.2, \quad \text{dla } t < 0. \quad (9.2)$$

Jest to tzw. *równanie logistyczne z opóźnieniem*. Dodatni współczynnik  $a$  jest bardzo istotnym parametrem. Okazuje się bowiem, że gdy  $a < \pi/2$ , to — dla dostatecznie długich czasów — rozwiązania (9.1) stabilizują się na poziomie  $a$ , natomiast dla  $a > \pi/2$  rozwiązania wpadają w niegasnące, regularne oscylacje. Więcej na temat logistycznego równania z opóźnieniem można dowiedzieć się z wykładu [Modelowanie matematyczne w biologii i medycynie](#).

Zacznijmy od zapisania skryptu, rozwiązującego nasze równanie dla wartości parametru  $a$  równych 0.1,  $\pi/2 - 0.1$ ,  $\pi/2 + .01$ , na przedziale czasowym długości  $T = 100$  (dostatecznie długim, byśmy zaobserwowali *prawdopodobną tendencję* rozwiązania).

```
global a;  
  
lag = 1;  
T = 100;
```

```

% parametry solvera
opts = odeset ('Stats', 'on', ...
    'RelTol', 1e-4, ... % tolerancja błędu względnego
    'AbsTol', 1e-4, ... % tolerancja błędu bezwzględnego
    'InitialStep', 1e-5, ... % początkowy krok schematu
    'MaxStep', 1e-2); % przy okazji steruje liczbą punktów probkowania rozwiązania

filename = 'logdel';

for a = [0.1, pi/2 - 0.1, pi/2 + 0.1]

    sol = ode23d(@logdelf, [0,T], 0.2, lag, 0.2, opts);
    t = sol.x; Y = sol.y; sol.stats
    Ylag = interp1(t,Y,t-lag,'cubic'); % rozwiązanie opóźnione
    Ydot = logdelf(t,Y,Ylag); % pochodna rozwiązania

    plot(t,Y);
    xlabel('t'); ylabel('y(t)'); title('Rozwiązanie_y(t)');
    print('-depsc', strcat(filename, '-a-', ...
        strrep(num2str(a,'%g'),' ','p'), '.eps') )

    pause(1);

    plot(Y,Ydot);
    xlabel('y(t)'); ylabel('y''(t)'); title('Wykres_fazowy');
    print('-depsc', strcat(filename, '-a-', ...
        strrep(num2str(a,'%g'),' ','p'), '-phase.eps') )

    pause(1);

end

```

Kluczowa jest linijka

```
sol = ode23d(@logdelf, [0,T], 0.2, lag, 0.2, opts);
```

- która w rzeczywistości *rozwiązuje* nasze równanie. Parametrami funkcji `ode23d` są kolejno:
- funkcja określająca prawą stronę<sup>1</sup> (u nas jest to `logdelf`), do której parametr `a` przekazujemy z przyzwyczajenia za pomocą zmiennej globalnej<sup>2</sup>,
  - przedział czasowy, na którym wyznaczamy rozwiązanie, tutaj:  $[0, T]$ ,
  - warunek początkowy,
  - wartości opóźnień (u nas jest tylko jedno, bo równanie jest skalarne) `lag = 1`), no i ewentualnie
  - struktura, zmieniająca domyślne parametry pracy *solvera*.

```

function ydot = logdelf(t,y,ylag)
global a;
ydot = (a - ylag).*y;
end

```

W odróżnieniu od `lsode`, `ode23d` zwraca wartości rozwiązania w przez siebie wyznaczonych punktach przedziału  $[0, T]$ . Aby więc określić jego wartość w zadanych przez nas punktach,

<sup>1</sup> Funkcja *musi* być przekazana jako wskaźnik, a nie jako nazwa!

<sup>2</sup> W rzeczywistości funkcje z pakietu `OdePkg` umożliwiają wygodne przekazywanie listy dodatkowych parametrów do wnętrza funkcji prawej strony.



```
t = linspace(0,T,10*T);
```

musimy interpolować wartości; warto także wcześniej zadać w opcjach maksymalną długość kroku, która zagwarantuje nam wystarczającą rozdzielczość próbkowania; dla danych opóźnionych, a następnie wyznaczyć

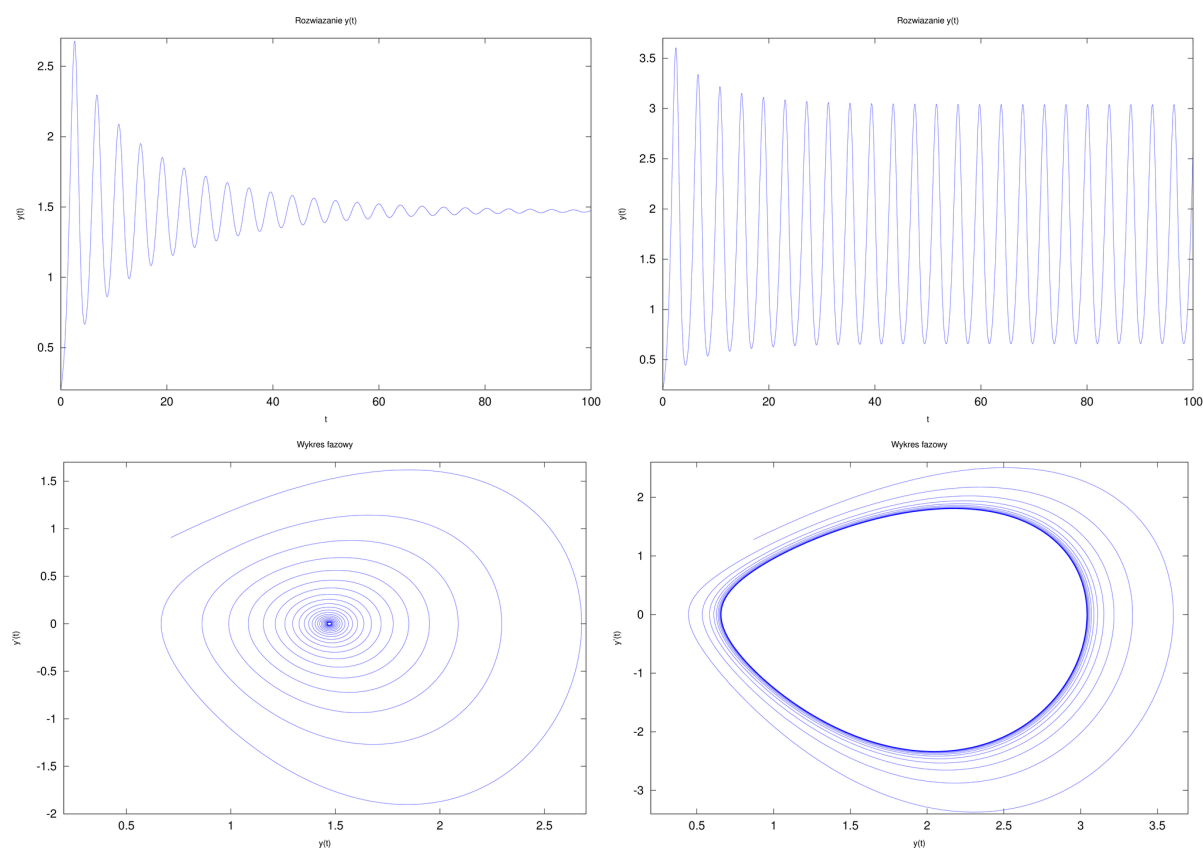
```
Ylag = interp1(t,Y,t-lag,'cubic'); % rozwiązanie opóźnione
Ydot = logdelf(t,Y,Ylag); % pochodna rozwiązania
```

która zwraca nam wartości obliczonego przybliżenia rozwiązania (i jego pochodnej!) w punktach  $t$ . Potem wystarczy je tylko narysować, komendami

```
plot(t, Y);
plot(Y, Ydot);
```

— najpierw rozwiązanie w zależności od czasu, potem — wykres fazowy.

Wyniki, które, przynajmniej w sposób jakościowy, zgadzają się z przewidywaniami teoretycznymi, przedstawia rysunek 9.1.



Rysunek 9.1. Wykresy rozwiązania (na górze) i wykresy fazowe (na dole) rozwiązań równania logistycznego z opóźnieniem dla parametru  $a$  równego  $1.4708 < \pi/2$  (po lewej) i  $1.6708 > \pi/2$  po prawej. Rozwiązania po lewej stronie stabilizują się wokół  $a$ , natomiast po prawej — oscylują niegasnąco wokół  $a$ .

## 9.2. Weryfikacja wyników

Aby набrać więcej przekonania do ilościowych wyników, zobaczmy, czy stukrotne zmniejszenie parametrów tolerancji błędu spowoduje wyraźną zmianę uzyskanych wartości rozwiązania. Sprawdzenie przeprowadzimy dla parametru  $a = \pi/2 + 0.1$ .

Najpierw wyznaczmy w standardowy sposób wartości rozwiązania w interesujących nas punktach:

```
a = pi/2 + 0.1;

opts = odeset ('Stats', 'on', ...
  'RelTol', 1e-4, ... % tolerancja błędu względnego
  'AbsTol', 1e-4, ... % tolerancja błędu bezwzględnego
  'InitialStep', 1e-5, ... % początkowy krok schematu
  'MaxStep', 1e-2); % przy okazji steruje liczbą punktów próbkowania

sol = ode23d(@logdelf, [0,T], 0.2, lag, 0.2, opts);
t = sol.x; Y = sol.y; sol.stats
```

a następnie powtórzmy obliczenia dla ostrzejszych kryteriów:

```
opts = odeset ('Stats', 'on', ...
  'RelTol', 1e-6, ... % tolerancja błędu względnego
  'AbsTol', 1e-6, ... % tolerancja błędu bezwzględnego
  'InitialStep', 1e-7, ... % początkowy krok schematu
  'MaxStep', 1e-2); % przy okazji steruje liczbą punktów próbkowania

sol = ode23d(@logdelf, [0,T], 0.2, lag, 0.2, opts);
tref = sol.x; Yref = sol.y; sol.stats
```

Jednak kłopot w tym, że — w przeciwieństwie do `lsode` — funkcja `ode23d` zwraca rozwiązanie nie we wskazanych przez nas wartościach  $t$ , ale raczej zwraca wszystkie wyznaczone przez siebie wartości, co oznacza, że dla różnych parametrów wywołania *solvera* prawie na pewno dostaniemy inny zestaw `sol.x`.

Dlatego musimy najpierw sprowadzić wyznaczone wartości do wspólnego zestawu chwil czasowych  $t_0, \dots, t_N$ . Wydaje się rozsądne, by za ten bazowy zestaw wziąć po prostu  $t$  — wartości wyznaczone przy mniej restrykcyjnych parametrach pracy *solvera*. Następnie, musimy *interpolować*<sup>3</sup> wartości  $(tref, Yref)$  na  $t$ , do czego idealnie nadaje się funkcja **interp1**.

```
Yleft = interp1(tref, Yref, t);
```

Następnie sprawdzamy, jak bardzo różnią się rozwiązania  $Y$  i  $Yref$ :

```
octave:> norm(Yleft-Y, Inf)
ans = 4.8362e-06
octave:> semilogy(abs(Yleft-Y));
```

A więc maksymalna różnica pomiędzy rozwiązaniami wynosi jedynie około  $10^{-6}$ , co jest raczej uspokajającą wiadomością (choć należałoby jeszcze wykluczyć na przykład zjawisko aliasingu, które mocno dało się nam we znaki, gdy rozwiązywaliśmy zadanie o transformatorze, z rozdziału 6).

**Ćwiczenie 9.1.** Potwierdź eksperymentalnie, że właśnie dla  $a = \pi/2$  następuje zmiana długoterminowego charakteru rozwiązań, z gasnących do zera, na oscylacyjne.

<sup>3</sup> To może wprowadzić dodatkowe zaburzenia w wynikach, więc musimy być czujni!

**Ćwiczenie 9.2.** Napisz funkcję, pozwalającą rozwiązać równanie logistyczne z opóźnieniem o zadanych parametrach i jednocześnie od razu zweryfikować w opisany powyżej sposób jakość rozwiązania. Zadbaj o komunikaty i wizualizacje dla niecierpliwego użytkownika. Następnie sprawdź pozostałe uzyskane powyżej wyniki.

**Ćwiczenie 9.3.** Napisz skrypt Octave lub MATLABa, rozwiązujący inną wersję równania logistycznego z opóźnieniem, pochodzącą z pewnego modelu wzrostu nowotworu u myszy [5]:

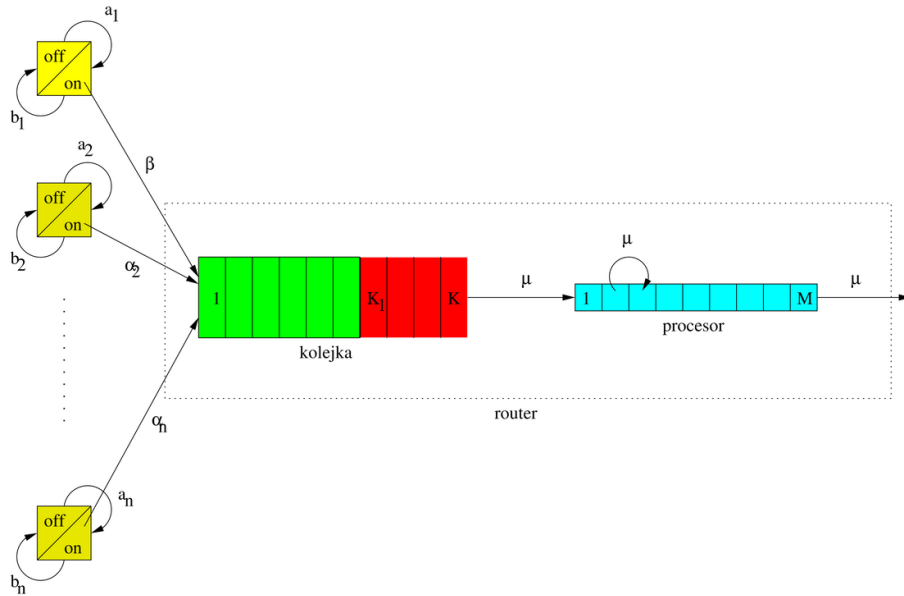
$$N'(t) = rN(t - \tau) \left( 1 - \frac{N(t - \tau)}{K} \right).$$

Dodatknie współczynniki  $r$  i  $K$  są parametrami modelu. Zweryfikuj jakość uzyskanych aproksymacji rozwiązania i jego pochodnej w przypadku, gdy  $r = 4$ ,  $K = 2$ , a opóźnienie wynosi  $\tau = 0.7$ . Jako funkcję początkową przyjmij stałą równą  $N_0(t) = 1.2$ .

## 10. Zadanie telekomunikacyjne

Rozważmy następujące zagadnienie, które pojawia się w analizie sieci telekomunikacyjnych<sup>1</sup>. Otóż przypuśćmy, że mamy  $n$  źródeł pakietów, które podłączone są do routera, który składa się z systemu kolejkującego pakiety i procesora, który je przetwarza. Źródła mogą w danej chwili być albo włączone, albo wyłączone, a wysyłać pakiety mogą oczywiście tylko wtedy, gdy są włączone.

Przyjmujemy, że  $i$ -te źródło przechodzi ze stanu wyłączzonego w stan włączony ze średnią częstością  $a_i$ , a ze stanu włączonego do wyłączzonego — z częstością  $b_i$ . Znaczą to na przykład, że średnio co  $1/a_i$  jednostek czasu źródło, które jest wyłączone, przechodzi do stanu włączonego. Gdy  $i$ -te źródło jest włączone, może wysłać pakiet danych do routera, który je kolejkuje i następnie, jeden po drugim, przetwarza w procesie składającym się z  $M$  etapów pośrednich.



Rysunek 10.1. Schemat sieci telekomunikacyjnej. Po lewej źródła pakietów (pierwsze jest uprzywilejowane), po środku kolejka długości  $K$ , po prawej procesor z  $M$ -etapowym przetwarzaniem.

Zakładamy, że średnia szybkość przetwarzania pakietu w procesorze wynosi  $\mu$ , czyli średnio co  $1/\mu$  jednostek czasu pakiet przechodzi do kolejnego etapu przetwarzania. Jeśli w procesorze nie ma pakietu, z tą samą częstością może zostać pobrany do przetwarzania nowy pakiet z kolejki.

Kolejka ma pojemność  $K$  pakietów i jest typu FIFO (*first in, first out*). Nowe pakiety są wysyłane do kolejki z  $i$ -tego źródła z częstością  $\alpha_i$  (pod naturalnym warunkiem, że jest ono włączone). Dodatkowo przyjmujemy, że pierwsze źródło jest uprzywilejowane i gdy długość kolejki przekroczy pewien zadany próg bezpieczeństwa  $K_1 \in (0, K)$ , przyjmowane są pakiety

<sup>1</sup> Zadanie to przedstawił mi dr hab. Piotr Pokarowski z Uniwersytetu Warszawskiego. Wiele podobnych przykładów Czytelnik znajdzie w [29].

tylko z pierwszego źródła: pakiety z pozostałych źródeł są tracone. Oczywiście, gdy kolejka jest pełna, pakiety ze wszystkich źródeł są odrzucane i tracone.

Dla zmniejszenia liczby parametrów modelu przyjmujemy (niezbyt realistycznie), że  $a_1 = a_2 = \dots = a_n = a$  oraz  $b_1 = b_2 = \dots = b_n = b$ , natomiast  $\alpha_1 = \beta$  oraz  $\alpha_2 = \dots = \alpha_n = \alpha$ .

W danej chwili, stan takiego układu źródła–kolejka–router jest więc charakteryzowany przez trójkę liczb naturalnych:  $(s, q, r)$ , gdzie

- $s = 0, \dots, 2^n - 1$  oznacza stan źródeł (każde ze źródeł może być albo wyłączone (stan jednostkowy „0”) albo włączone (stan jednostkowy „1”); jeśli więc  $i$ -te źródło jest w jednostkowym stanie  $s_i \in \{0, 1\}$ , to kładziemy

$$s = \sum_{i=1}^n s_i 2^{i-1},$$

czyli  $s_i$  są cyframi zapisu  $s$  w układzie dwójkowym:  $s = (s_n s_{n-1} \dots s_2 s_1)_2$ ;

- $q = 0, \dots, K - 1$  oznacza długość kolejki;
- $r = 0, \dots, M - 1$  oznacza etap przetwarzania pakietu w procesorze.

Alternatywnie, można byłoby rozpatrywać nie trójwymiarową, ale  $(n+2)$ -wymiarową przestrzeń stanów, gdzie pierwsze  $n$  stanów mogłoby przyjmować tylko wartości 0 albo 1. Numerując wszystkie  $N = 2^n K M$  stanów możemy utworzyć macierz przejścia  $W = (w_{ij})_{i,j=1}^N$ , w której element  $w_{ij}$  odpowiada częstości przejścia ze stanu  $i$  do stanu  $j$ ; ponadto kładziemy  $w_{ii} = -\sum_{j \neq i} w_{ij}$ . Przyjmując, że modelowany przez nas układ „nie ma pamięci”, to znaczy: stan chwili następnej zależy jedynie od stanu w chwili bieżącej, konkludujemy, że mamy do czynienia z łańcuchem Markowa z ciągłym czasem i dyskretną przestrzenią stanów.

Zgodnie z regułami omówionymi na początku, mogą zajść następujące zdarzenia:

- któreś ze źródeł się włączy lub wyłączy,
- kolejka powiększy się o kolejny pakiet,
- zostanie ukończony kolejny etap przetwarzania pakietu w procesorze,
- procesor pobierze z kolejki nowy pakiet.

W naszym modelu założymy, że w jednej chwili możliwe jest wystąpienie *tylko jednego* z nich.

Zatem jeśli zmiana stanu dotyczy źródeł, to w danym momencie może zmienić się stan tylko jednego z nich: albo przestanie nadawać, albo przestanie wysyłać (ewentualnie nie zmienia stanu). Dlatego jeśli w zapisie binarnym  $s = (s_N s_{N-1} \dots s_1)_2$ , to  $s$  może przyjąć tylko takie nowe wartości, które różnią się dokładnie jedną cyfrą zapisu binarnego.

Dalej, może zwiększyć się długość kolejki. Jeśli jest włączonych  $p$  źródeł, w tym uprzywilejowane, to długości kolejki wzrośnie o 1 z częstością  $\beta + (p-1)\alpha$  (o ile nie są przekroczone wartości graniczne, o których była mowa na samym początku). Oznacza to przejście ze stanu  $(s, k, r)$  do stanu  $(s, k+1, r)$  ze wspomnianą wyżej częstością. Gdy kolejka przekroczy długość  $K_1$ , wtedy dopuszczane są tylko pakiety ze źródła uprzywilejowanego: gdy jest ono włączone, długości kolejki wzrośnie o 1 z częstością  $\beta$ .

W końcu, stan może zmienić się wskutek przetworzenia pakietu przez procesor. Jeśli router wypuści pakiet na zewnątrz, układ przechodzi ze stanu  $(s, q, M)$  do stanu  $(s, q, 0)$ . Jeśli router jest wolny (czyli *już jest* w stanie  $(s, q, 0)$ ), z kolejki może zostać pobrany nowy pakiet (czyli przejdziemy do stanu  $(s, q-1, 1)$ ; jeśli kolejka jest pusta, nie stanie się nic). Wreszcie, pakiet przetwarzany przez procesor może przesunąć się do kolejnego etapu przetwarzania, to znaczy — do stanu  $(s, q, r+1)$ . Wszystkie te zależne od procesora zmiany stanu odbywają się z częstością  $\mu$ .

W zastosowaniach interesujący jest stan stacjonarny  $\pi$  naszego łańcucha, to znaczy stan, w jakim powinien ustabilizować się nasz układ po bardzo długim czasie. Współrzędna  $\pi_i$  odpowiada średniemu procentowi czasu, w którym nasz system znajduje się w stanie o numerze  $i$ . Możemy więc być zainteresowani takim doбором parametrów sieci (tzn. częstości lub np.

długości kolejki), by np. nie tracić (średnio) zbyt wielu pakietów. Stan stacjonarny  $\pi$  jest dany równaniem [29]

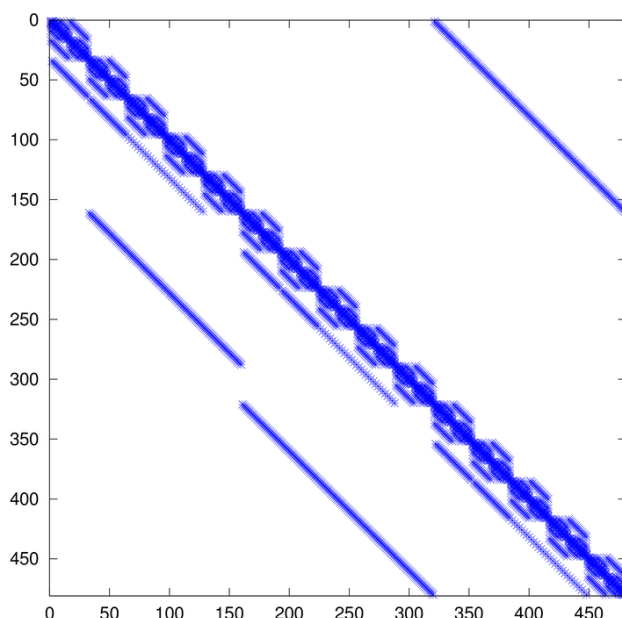
$$W^T \pi = 0, \quad (10.1)$$

przy czym  $\pi_i \geq 0$  oraz  $\sum_i \pi_i = 1$ .

I to jest właśnie zadanie z jakim się zmierzmy: spróbujemy znaleźć wektor  $\pi$ , spełniający (10.1).

### 10.1. Konstrukcja macierzy przejścia

Jak widzieliśmy powyżej, w naszym przypadku, z danego stanu układ może przejść jedynie do *kilku* — dokładniej: do  $n+1+1$  innych (lub pozostać w stanie niezmiennym). Znaczący to, że macierz  $W$  jest silnie rozrzedzona, co oznacza, że *musimy* ją reprezentować w postaci macierzy rzadkiej.



Rysunek 10.2. Macierz przejścia dla  $n = 5, K = 5, M = 3, K_1 = 3..$

Wszystkich stanów jest  $N = 2^n K M$  i, aby skonstruować macierz  $W$ , musimy je najpierw ponumerować według jakiejś reguły<sup>2</sup>. Ale jak? Narzuca się jedno z sześciu naturalnych rozwiązań<sup>3</sup>:

**sqr**  $i = s + 2^n q + 2^n K r$

**srq**  $i = s + K r + 2^n K q$

**rsq**  $i = r + M s + 2^n M q$

**rqs**  $i = r + M q + M K s$

**qsr**  $i = q + K s + 2^n K r$

**qrs**  $i = q + K r + M K s$

<sup>2</sup> Wybór reguły może mieć znaczenie dla efektywności działania metody rozwiązywania układu równań (10.1)!

<sup>3</sup> Na razie przyjmijmy, że indeksujemy od zera.

Pytanie o to, który z nich okaże się najlepszy (i czy możliwy jest *jeszcze lepszy* sposób) zostawimy na później. Na razie przyjmiemy wybór reguły silnych powiązań, dzięki czemu uzyskamy macierz o możliwie zagęszczonych wokół diagonal elementach. Ponieważ stany źródeł mogą zmieniać się aż na  $2^n$  sposobów, wybierzemy je jako najszybciej zmieniającą się współrzędną trójki  $(s, q, r)$  i zdecydujemy się na zakodowanie numeru przez

$$i = s + 2^n q + 2^n K r.$$

W praktycznej implementacji będziemy musieli jeszcze pamiętać o tym, że w Octave i MATLABie elementy wektora musimy indeksować od „1”, a w języku C — od „0”.

### 10.1.1. Implementacja w małej skali: Octave

Jak widać z powyższego — i tak właśnie bardzo często zdarza się w najrozmaitszych zadaniach obliczeniowych matematyki stosowanej — nasz problem *skaluje się*, to znaczy można go postawić zarówno dla niewielkiej liczby niewiadomych (na przykład dla  $n = 2$  i  $K = M = 4$ ), jak też dla *bardzo dużej* liczby parametrów (wystarczy wziąć dostatecznie duże  $n$ : już dla  $n = 64$  liczba samych stanów źródeł,  $2^{64}$ , przekroczy możliwości adresowania współczesnych procesorów!).

Możemy więc na próbę, dla testów i wstępnych obserwacji spróbować wykorzystać środowisko Octave dla małych  $n, K, M$ . Utworzone przez nas narzędzia dla zadania małej skali mogą przydać się np. do pre- lub postprocessingu danych lub wyników zadania w dużej skali.

Naszą macierz  $W$  będziemy tworzyć wierszami, według schematu:

```
for r=0:M-1
  for q = 0:K-1
    for s = 0:2^n-1
      % wyznacz numer stanu i odpowiadający trójce (s,q,r)
      % wyznacz częstości w.ij przejścia od stanu i=(s,q,r) do innego stanu j
    end
  end
end
```

Alternatywnie, moglibyśmy użyć jednej pętli po wszystkich numerach stanów:

```
for i=1:N
  % wyznacz trójkę (s,q,r) odpowiadającą stanowi i
  % wyznacz częstości przejścia od stanu (s,q,r) do innego stanu j
end
```

Będzie więc nam potrzebna funkcja, która danemu stanowi, reprezentowanemu przez trójkę  $(s, q, r)$ , przyporządkowuje jego numer kolejny, na przykład

```
function i = triple2state(x,z,p)
% nadaje numer "i" stanowi reprezentowanemu trojka [x(1),x(2),x(3)]
% zakładamy, że x(1) = 0..2^n-1, x(2) = 0..K-1, x(3) = 0..M-1,
% gdzie z = [2^n K M]
% p jest sposobem numerowania, np. aby dostać sposób "sqr", bierzemy p = [1 2 3],
% dla sposobu "qrs" kładziemy p = [2 3 1], itp.

% schemat "p(1)-p(2)-p(3)"

i = 1 + x(p(1)) + z(p(1)) * ( x(p(2)) + z(p(2)) * x(p(3)) );
% stany powinniśmy numerować od jedynki, bo numer stanu
% będzie indeksem w macierzy!
```

```
end
```

**Ćwiczenie 10.1.** Zaprogramuj w Octave funkcję odwrotną do `triple2state`, to znaczy — wyznaczającą trójkę  $(s, q, r)$  na podstawie numeru stanu.

*Rozwiązanie.* Poniżej przykładowy kod.

```
function x = state2triple(k,z,p)
% zamienia numer stanu k na trojke opisujaca stan, [x(1),x(2),x(3)]
% zakladamy, ze x(1) = 0..2^n-1, x(2) = 0..K-1, x(3) = 0..M-1
% gdzie z = [2^n K M]

x = [-1,-1,-1];
k = k-1;

q = fix(k/z(p(1)));
a = k - q*z(p(1));
c = fix(q/z(p(2)));
b = q - c*z(p(2));

x(p) = [a,b,c];
end
```

### Praca z numerami stanów źródeł

Ze względu na to, że stany kolejnych źródeł kodujemy ustawiając konkretne *bity* w liczbie  $s$ , musimy nauczyć się pracować z pojedynczymi bitami w danej zmiennej całkowitoliczbowej. Poniżej przypomnijmy więc podstawowe informacje na ten temat:

**Sprawdzenie bitu** Aby sprawdzić, czy  $i$ -ty bit liczby  $s = (s_N s_{N-1} \dots s_1)_2$  jest równy jeden, musimy wykonać bitową operację AND na  $s$  i na  $m = (0 \dots 0 \underbrace{1}_i 0 \dots 0)_2$ . Wynik nie jest

zerem wtedy i tylko wtedy, gdy  $s_i = 1$ .

**Zapalenie bitu** Aby ustawić wartość  $i$ -tego bitu liczby  $s$  na 1, należy wykonać bitową operację OR na  $s$  i  $m$ .

**Zgaszenie bitu** Aby ustawić wartość  $i$ -tego bitu liczby  $s$  na 0, należy wykonać bitową operację AND na  $s$  i NOT  $m$ , czyli na  $s$  i na  $(1 \dots 1 \underbrace{0}_i 1 \dots 1)_2$ .

**Przełączenie bitu** Aby zmienić wartość  $i$ -tego bitu liczby  $s$  na inną (czyli z 0 na 1 i na odwrót, możemy wykorzystać gotowy operator w języku C, w przeciwnym razie musimy skorzystać z opisanych powyżej funkcji.

Wszystkie powyższe operacje można w prosty sposób przeprowadzić w języku C i w Octave/MATLABie, ale należy pamiętać, że w C bity indeksujemy od zera.

Operacja na $i$ -tym bicie	C ( $i = 0, \dots$ )	Octave ( $i = 1, \dots$ )
Czy zapalony	$z = s \& (1 << i)$	$z = \text{bitget}(s, i)$
Zapalenie	$S = s \mid (1 << i)$	$S = \text{bitset}(s, i, 1)$
Zgaszenie	$S = s \& \sim(1 << i)$	$S = \text{bitset}(s, i, 0)$
Przełączenie	$S = s \wedge (1 << i)$	$S = \text{bitset}(s, i, \sim \text{bitget}(s, i))$

Znaczy to, że jeśli chcemy znaleźć wszystkie możliwe stany osiągalne z danego stanu  $s$  i nadać wartości częstości przejścia źródeł ze stanu  $s$  do nowego stanu, musimy wykonać pętlę

```
S = triple2state([s,q,r],z,ord); % numer biezacego stanu
```



```

%%%%%%%%%%%% zmiana stanu z powodu zmiany stanu zrodla

for i = 1:n
    bit = bitget(s,i);
    sn = bitset(s,i,~bit); % zmienia i–ty bit na przeciwny: to nowy numer stanu zrodla
    SN = triple2state([sn,q,r],z,ord); % numer nowego stanu zrodla

    if bit == 0
        W(S,SN) = a; % przejście od wyłączonego do włączonego
    else
        W(S,SN) = b; % przejście od włączonego do wyłączonego
    end
end
end

```

Z kolei zmianę stanu kolejki realizowałby następujący kod:

```

S = triple2state(s,q,r); % numer bieżącego stanu łańcucha

%%%%%%%%%%%% zmiana stanu z powodu zmiany stanu kolejki

if q < (K-1) % tylko wtedy można wydłużyć kolejkę, w przeciwnym razie wszystkie pakiety są odrzucane
    SN = triple2state([s,q+1,r],z,ord); % nowy stan: przyjęto pakiet do kolejki
    bs = bitsum(s); % ile jest włączonych
    b0 = bitget(s,1); % czy zerowy jest włączony
    if bs > 0 % cokolwiek nadaje

        W(S,SN) = 0;

        if b0 > 0 % specjalne traktowanie pierwszego zrodla
            W(S,SN) = beta;
        end
        if q < (K1-1) % pakiety z innych są odrzucane jeśli kolejka ma długość K1 lub większą
            W(S,SN) = W(S,SN) + (bs-b0)*alpha;
        end
    end
end
end

```

Wreszcie, może zmienić się stan procesora:

```

S = triple2state(s,q,r); % numer bieżącego stanu łańcucha
%%%%%%%%%%%% zmiana stanu z powodu zmiany stanu procesora

switch r
case (M-1)
    SN = triple2state([s,q,0],z,ord);
    W(S,SN) = mu;
case 0
    if (q > 0)
        SN = triple2state([s,q-1,1],z,ord);
        W(S,SN) = mu;
    end
otherwise
    SN = triple2state([s,q,r+1],z,ord);
    W(S,SN) = mu;
end
end

```

Powyżej określone operacje traktowały  $W$  jak macierz gęstą. W realnej implementacji wybierzemy format macierzy rzadkiej, dlatego ostateczna funkcja generująca macierz przejścia  $W$  będzie postaci:

```
function spW = sptelecomtrans(n,K,M,K1,rate)
% create (sparse) transition rate matrix spW
%
% queue length 0..K-1
% source states 0..2^n-1
% processing phases 0..M-1
% queue threshold K1

if nargin < 5
    rate = [1 1 1 1 1];
```



To tylko fragment kodu źródłowego Octave. Więcej na <http://mst.mimuw.edu.pl/lecture.php?lecture=ona&part=Ch10>.

Dobrze, że zaczynamy od Octave; tu najprościej testować implementacje, np. moglibyśmy szybko na kilku przykładach sprawdzić, czy nasze procedury: przypisywania częstości przejścia oraz numerowania stanów działają poprawnie. Nie sposób przecenić czasu (i nerwów) jakie dzięki tym wstępnym testom można zaoszczędzić.

**Ćwiczenie 10.2.** Sprawdź na małych przykładach, czy otrzymujesz poprawne macierze przejścia. Na przykład<sup>4</sup>, dla  $n = M = K = 1$  i  $a = 1, b = 2$  (co prawda nie można nadać im znaczenia częstości, ale to bez znaczenia dla testów) powinieneś otrzymać macierz

$$W = \begin{pmatrix} -1 & 1 \\ 2 & -2 \end{pmatrix}.$$

Pamiętaj, wyrazy na diagonalu  $W$  są określone zależnością  $w_{ii} = -\sum_{j \neq i} w_{ij}$ !

## 10.2. Wyznaczenie stanu stacjonarnego

Wydaje się, że nasze zadanie polega na znalezieniu jądra macierzy  $W^T$ , a następnie wyluskaniu zeń (i unormowaniu) odpowiedniego wektora. Rzeczywiście, w Octave jest funkcja

```
W = sptelecomtrans(n,K,M,K1,[a,b,alpha,beta,mu]);
X = null(W');
```

która wyznacza jądro zadanej macierzy. Jeśli więc jądro jest jednowymiarowe (tego nie wiemy), a wszystkie współrzędne są dodatnie (to musimy sprawdzić i ewentualnie sobie zagwarantować), to

```
X = X*sign(X(1)); % gwarancja znaku
if all(X>0) % sprawdzenie dodatniości
    Pi = X/norm(X,1);
else
    Pi = NaN(size(X));
```

<sup>4</sup> Doświadczenie uczy, że wykonanie *tylko tego jednego* testu to za mało, by wychwycić wszystkie dotąd popełnione pomyłki.

```
end
residW = norm(W'*Pi,Inf)
```

Niestety, wyznaczanie jądra macierzy generalnie przy dużych  $N$  może być bardzo kosztowne obliczeniowo. W dodatku, jego implementacja w Octave wykorzystuje słuszną, ale drogą metodę rozkładu SVD macierzy. Dzięki temu, że  $W$  jest rozrzedzona i ma kilka innych ciekawych matematycznych własności, będziemy mogli obejść to ograniczenie.

**Ćwiczenie 10.3.** Zaimplementuj metodę rozwiązywania zadania wykorzystującą `null` i zbadaj, jakie maksymalnie mogą być  $n$ ,  $K$ ,  $M$  dla których otrzymasz sensowne rozwiązanie w sensownym czasie.

*Wskazówka.* Generalnie, wszystkie potrzebne komendy zostały wcześniej przedstawione, ale powinniśmy jeszcze zabezpieczyć się przed sytuacją, w której `null` zwróci więcej niż jeden wektor. Taką sytuację należy potraktować jako indykator, że zadanie nie jest dobrze postawione i zgłosić przynajmniej ostrzeżenie:

```
X = null(W');
if size(X,2) > 1
    warning('Nonunique_solution');
    X = X(:,end); % wybieramy jakikolwiek (zwykle ostatni odpowiada wartosci
end                    szczególnej najblizszej zera)
```

### 10.2.1. Wybór właściwej metody

Najwyższy czas, by **sięgnąć do literatury**. W monografii [29], dotyczącej numerycznego wyznaczania różnych cech — także stanów stacjonarnych — łańcuchów Markowa jest omówiona metoda, w której poprzez *usunięcie* jednego wiersza i jednej kolumny z macierzy  $W$  można zadanie sprowadzić do zwykłego zadania rozwiązywania układu równań liniowych z macierzą *nieosobliwą*. Zapewne dla średnich rozmiarów macierzy  $W$  to byłaby całkiem dobra alternatywa — wszak  $W$  jest macierzą rozrzedzoną i taką pozostanie po usunięciu wiersza i kolumny — ale musielibyśmy nieco obawiać się postępującego *wypełnienia* bloków czynników rozkładu LU macierzy.

Inny sposób podejścia do zadania wyznaczenia stanu stacjonarnego łańcucha Markowa (por. [28]) to wykonanie tzw. odwrotnej metody potęgowej na macierzy (raz tylko sfaktoryzowanej!)  $W^T - \epsilon I$ . Jest to metoda iteracyjna rozwiązywania zagadnienia własnego, której zasadniczym składnikiem jest wielokrotne rozwiązywanie układu równań postaci  $W^T - \epsilon I$ . Aby więc metoda była efektywna, musimy umieć wyznaczyć rozkład macierzy  $W^T - \epsilon I$  na prostsze składniki, najlepiej — wykorzystując fakt, że macierz  $W^T$  jest mocno rozrzedzona. Więcej na temat metody potęgowej i odwrotnej metody potęgowej można przeczytać w wykładzie z [Matematyki Obliczeniowej II](#). Jednak w naszym przypadku domyślnie stosowana w Octave metoda rozkładu macierzy rozrzedzonych zabiera zbyt wiele czasu i jest mało efektywna, gdy  $N$  jest duże.

Na szczęście, jest *jeszcze jedna* droga, wykorzystująca *dodatkowe* właściwości naszego zadania. (Skąd my to znamy?...). Skoro mamy do czynienia z łańcuchem Markowa, to możemy zapisać zadanie w równoważnej postaci z macierzą *stochastyczną*<sup>5</sup>  $P$ ,

$$P\pi = \pi,$$

gdzie  $P = \delta W^T + I$ , przy czym  $\delta = 0.99 / \max_i |w_{ii}|$  (dowód równoważności i stochastyczności  $P$  zostawiamy Czytelnikowi jako niezobowiązujące ćwiczonko, zob. [28]). Zatem poszukiwany

<sup>5</sup> Czyli taką, której elementy są nieujemne i w każdej kolumnie sumują się do jedności.

wektor  $\pi$  jest wektorem własnym macierzy (kolumnami) stochastycznej  $P$ , odpowiadającym wartości własnej równej 1. Jest to znacząca informacja, gdyż to pociąga za sobą, że:

- istnieje wektor własny odpowiadający wartości własnej 1 (rzeczywiście, *lewostronny* wektor własny dla 1 to wektor  $(1, \dots, 1)$ ).
- wartość własna 1 jest *dominującą* wartością własną (rzeczywiście,  $|\lambda(P)| \leq \|P\|_1 = 1$ ) i nie ma innych wartości własnych o module równym 1 (wynika to z twierdzenia Gerszgorina).

Ponadto nasza macierz  $P$  jest nieredukowalna (w języku łańcuchów Markowa, łańcuch jest nieprzywiedlny: każdy stan jest osiągalny z każdego innego, w skończonej liczbie kroków), więc na mocy twierdzenia Perrona–Frobeniusa [3, 29] wartość własna 1 macierzy  $P$  jest pojedyncza, a odpowiada jej wektor własny  $\pi$  o dodatnich współrzędnych.

Te wszystkie fakty oznaczają, że — zamiast kosztownego SVD — możemy spróbować użyć na przykład *metody potęgowej* wyznaczania dominującej wartości własnej macierzy  $P$ !

Należy pamiętać, że metoda potęgowa, oprócz ewidentnej zalety (jest metodą iteracyjną, więc można obliczenia przerwać w bardziej dogodnym momencie, a iteracja opiera się na bardzo tanim mnożeniu,  $\pi^{(k+1)} = P\pi^{(k)}$ , przez macierz rozrzedzoną  $P$ ; ma też minimalne wymagania pamięciowe, co istotne przy dużych  $n$ ), ma też wady: jeśli druga co do wielkości wartość własna macierzy,  $\lambda_2$ , jest bliska dominującej —  $|\lambda_2| \approx 1$  — zbieżność metody może być bardzo wolna.

W Octave możemy wyznaczyć dominującą wartość własną na dwa sposoby: zaimplementować wprost metodę potęgową, albo wykorzystać funkcję `eigs`, wykorzystującą bardziej zaawansowane techniki wyznaczania ekstremalnych wartości własnych. Poniżej wybieramy właśnie ten drugi wariant:

```
opts.maxit = MAXIT;
opts.tol = TOL;
tic; [V, D, info] = eigs(spW, 6, 'lm', opts); toc
D = diag(D); [m i] = max(abs(D));
if (abs(m-1) > 1e1*eps)
    warning(['Dominant eigenvalue = ', num2str(m, '%e'), ' not equal to 1. Difference: ', num2str(abs(m-1))]);
end
Pi = V(:,i);
Pi = abs(Pi); % ustalamy znak Pi
Pi = Pi/sum(Pi);
```

Pozostaje jeszcze przerobić macierz  $W$  na macierz  $P$ ; w naszym przypadku zastąpimy funkcję `W = sptelecomtrans(..)` funkcją `[P, W] = sptelecom(..)`, wyznaczającą przede wszystkim macierz  $P$ , a macierz  $W$  — tylko opcjonalnie.

```
function [spP, spW] = sptelecom(n,K,M,K1,rate)
% create (sparse) transition probability matrix spP and (if requested)
% transition rate matrix spW
%
% queue length 0..K-1
% source states 0..2^n-1
% processing phases 0..M-1
% queue threshold K1

if nargin < 5
```



To tylko fragment kodu źródłowego Octave. Więcej na <http://mst.mimuw.edu.pl/lecture.php?lecture=ona&part=Ch10>.

Parametry zadania zapiszemy w prostym skrypcie Octave, o nazwie `telecomparam.m`:

```
a = 3.33;
b = 3.52;
alpha = 1;
beta = 2;
mu = 34e-1; % uwaga

n = 13;
K = 8;
M = 5;
K1 = 7;
MAXIT = 100;
TOL = 1e-6;
```

Aby wygenerować macierz  $P$ , wystarczy więc wydać polecenia:

```
telecomparam;
P = sptelecom(n,K,M,K1,[a,b,alpha,beta,mu]);
```

### 10.3. Zadanie w dużej skali

Ponieważ liczba wszystkich stanów  $N$  zależy eksponencjalnie od  $n$ , to gdy  $n$  jest większe niż kilka, generowanie samej macierzy  $P$  w Octave będzie trwało potwornie wolno: z powodu konieczności zinterpretowania wielokrotnie zagnieżdżonych pętli.

Dlatego, aby móc prowadzić obliczenia dla dużych  $N$ , powinniśmy przyspieszyć proces generowania macierzy  $P$ , implementując procedurę na przykład w języku C.

#### 10.3.1. Implementacja procedury generowania macierzy w C

W zasadzie wystarczy wprost przetłumaczyć tekst procedury z Octave na język C, pamiętając wszakże o następujących pułapkach, które czyhają na nas po drodze:

- tablice w C indeksujemy od zera, dlatego
  - licznik `nz` będziemy inicjalizować na  $-1$ ,
  - funkcja `triple2state` będzie zwracać  $s + N \cdot (q + K \cdot r)$ , a nie jak w Octave,  $1 + s + N \cdot (q + K \cdot r)$ ,
- bity w C numerujemy od zera, dlatego pętla po bitach źródeł powinna być dla  $i = 0, \dots, n-1$ :
 

```
for (i = 0; i < n; i++)
```

 a bitem źródła uprzywilejowanego źródła jest bit zerowy `b0 = bitget(s,0)`; funkcja `bitget(s,i)` powinna akceptować wartości  $i = 0, \dots, n-1$ ;
- w kodzie zakładamy, że `bitget()` zwraca zero lub 1, dlatego nie możemy użyć prostego `s & (1 << i)`, które może zwracać wartości większe od 1;

Ponadto, nie będziemy już generować (niepotrzebnej w tej chwili) macierzy częstości  $W$ , a tylko samą macierz  $P$ .

Warto zwrócić uwagę na fragment pozwalający prosto odczytać wszystkie parametry zadania zapisane w pliku Octave `telecomparam.m`.

Głównym celem programu `sptelecom.c` jest wyznaczenie macierzy rozrzedzonej  $P$  w formacie współrzędnych i zapisanie jej do pliku (aby można było następnie wczytać ją do Octave i potraktować `eigs`).

Tu kryje się jeszcze jedna pułapka. Zapisując macierz w formacie tekstowym, ryzykujemy utratę dokładności reprezentacji wartości elementów macierzy  $P$ . Dlatego, lepiej będzie zapisywać dane w formacie *binarnym* — i tak faktycznie zrobimy: patrz kod źródłowy programu w języku C.

```

/*
gcc -o sptelecom sptelecom.c -lm
time ./sptelecom < telecomparam.m
*/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define NPARAM 11 /* liczba parametrow do wczytania z pliku */

/* zmienne, do ktorych wczytamy parametry z pliku */
int n, N, K, M, K1;
double a, b, alpha, beta, mu;
int MAXIT; double TOL;

```



To tylko fragment kodu źródłowego w C. Więcej na <http://mst.mimuw.edu.pl/lecture.php?lecture=ona&part=Ch10>.

### 10.3.2. Rozwiązanie zadania w Octave

Mając macierz  $P$  w formacie współrzędnych, w postaci binarnej, musimy najpierw zmusić Octave do wczytania jej właśnie w tej postaci:

```

function [spW, dim] = sptelecomload(filename)
fp = fopen(filename, 'rb');
nz = fread(fp, 1, 'int');
IJ = fread(fp, [nz, 2], 'int');
IJ = IJ + 1; % indeksy nie od zera, ale od 1
dim = max(IJ(:, 1));
V = fread(fp, nz, 'double');
fclose(fp);

spW = sparse(IJ(:, 1), IJ(:, 2), V, dim, dim);
end

```

Teraz możemy już rozwiązać zadanie:

1. Wpisać poprawne wartości parametrów do pliku `telecomparam.m`
2. Wygenerować plik z macierzą  $P$ :

```

make sptelecom
./sptelecom < telecomparam.m

```

3. Wyznaczyć wektor  $\pi$ :

```

telecomparam;
system('make_cout.dat');

[P, dim] = sptelecomload('cout.dat');

opts.maxit = MAXIT;
opts.tol = TOL;
tic; [V, D, info] = eigs(P, 2, 'lm', opts); toc

```

```

D = diag(D); [m i] = max(abs(D));
if (abs(m-1) > 1e1*eps)
    warning(['Dominant_eigenvalue=', num2str(m, '%e'), ' _not_equal_to_1._Difference:', num2str(abs(m-1))]);
end
Pi = V(:,i);
Pi = abs(Pi);
Pi = Pi/sum(Pi);
residP = norm(Pi-P*Pi,Inf)
if (abs(residP) > 1e1*eps)
    warning(['Residual_||(P-I)*Pi||=', num2str(residP)]);
end

semilogy([0:K-1],sum(sum(reshape(Pi,[2^n,K,M]),3),1),'o-');
% semilogy([0:2^n-1],sum(sum(reshape(Pi,[2^n,K,M]),3),2),'o-');

file=fopen('moutx.dat','w');
fprintf(file, '%23.18e\n', Pi);
fclose(file);

```

Ponieważ Octave potrafi uruchamiać z poziomu sesji inne programy systemowe, wykorzystaliśmy to do automatycznego wygenerowania nowych danych, jak tylko zmieniły się parametry zapisane w pliku, lub zmianie uległ sam kod źródłowy.

**Ćwiczenie 10.4.** Która współrzędna  $\pi$  jest największa, gdy  $a = 3.33$ ,  $b = 3.52$ ,  $\alpha = 1$ ,  $\beta = 2$ ,  $\mu = 3.4$ ,  $n = 13$ ,  $K = 8$ ,  $M = 5$ ,  $K_1 = 7$ ? Sprawdź, jaki jest przewidywany procent czasu, w którym kolejka ma zadaną długość  $L$  z przedziału  $0, \dots, K$ . Z jakim prawdopodobieństwem będziemy tracić pakiety z nie-priorytetowych źródeł?

*Rozwiązanie.*

```

semilogy([0:K-1],sum(sum(reshape(Pi,[2^n,K,M]),3),1),'o-');

```

## 11. Równanie reakcji–dyfuzji

Ten wykład jest modyfikacją jednego z rozdziałów w [20], uwzględniającą demonstrację sposobu poprowadzenia wizualizacji rozwiązań.

Rozważmy proste nieliniowe zagadnienie ewolucyjne (zmienną  $t$  będziemy interpretować jako czas), określone na prostokącie  $\Omega = (0, a) \times (0, b)$ :

$$\begin{aligned} u_t &= \beta \Delta u + f(u) && \text{w } \Omega \times (0, T) \\ u &= 0 && \text{na } \partial\Omega \times (0, T), \\ u &= u_0 && \text{na } \Omega \times \{0\}. \end{aligned} \quad (11.1)$$

Niewiadomą jest funkcja  $u = u(x, y, t)$ . Zadana funkcja  $u_0(x, y)$  określa wartość rozwiązania w chwili początkowej  $t = 0$ . Symbolem  $\Delta$  oznaczamy dwuwymiarowy operator Laplace’a,

$$\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}.$$

Równanie (11.1) nazywamy równaniem *reakcji–dyfuzji*, gdyż jeśli  $u$  modelowałoby stężenie jakiejś substancji chemicznej, to  $\beta > 0$  byłoby jej współczynnikiem dyfuzji, a  $f$  — członem odpowiedzialnym za szybkość reakcji chemicznej, zależną od wartości  $u$ . Dla ustalenia uwagi, przyjmiemy dalej, że

$$f(u) = u - u^3.$$

Do rozwiązywania tego zadania zastosujemy tzw. metodę linii, w której (11.1) przybliżymy wielkim układem równań różniczkowych *zwyczajnych* względem czasu. Zaczniemy od samodzielnej dyskretyzacji po zmiennych przestrzennych  $x, y$ . Wybierzemy tu najmniej efektywną, ale za to koncepcyjnie najprostszą metodę dyskretyzacji, opartą na równomiernej siatce węzłów  $p_{ij} = (x_i, y_j) \in \Omega$ . Przyjmiemy  $x_i = ih_x, y_j = jh_y$ , gdzie  $h_x = a/(N_x + 1)$  jest krokiem siatki w kierunku  $x$  i analogicznie  $h_y = b/(N_y + 1)$ , zob. rysunek ??.

Oznaczając  $u_{ij}(t) = u(x_i, y_j, t)$ , a następnie zastępując pochodne przestrzenne ilorazami różnicowymi [15]

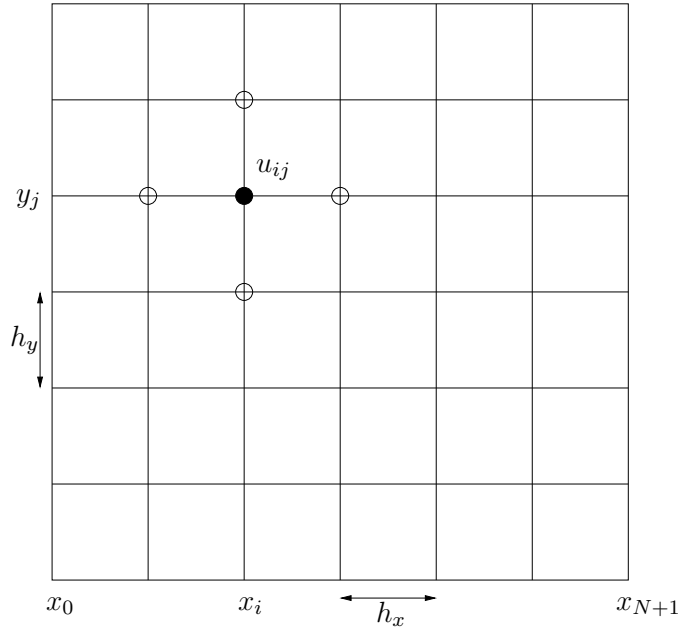
$$\Delta u_{ij} \approx \Delta_h u_{ij} = \frac{1}{h_x^2} (2u_{i,j} - u_{i+1,j} - u_{i-1,j}) + \frac{1}{h_y^2} (2u_{i,j} - u_{i,j+1} - u_{i,j-1}), \quad (11.2)$$

przybliżamy (11.1) układem równań różniczkowych *zwyczajnych*:

$$\frac{d}{dt} U(t) = \beta \Delta_h (U(t)) + f(U(t)). \quad (11.3)$$

Macierz  $U$  jest rozmiaru  $N_x \times N_y$  o elementach będących funkcjami czasu:  $U(t) = (u_{ij}(t))$  dla  $1 \leq i \leq N_x, 1 \leq j \leq N_y$ , a operator  $\Delta_h$  odpowiada macierzy dyskretnego dwuwymiarowego





Rysunek 11.1. Regularna siatka na  $\Omega$ .

laplasjanu (11.2). W węzłach brzegowych,  $U$  jest oczywiście równe zero, z warunków zadania, więc nie musimy go wyznaczać.

Do uzyskanego układu równań różniczkowych zwyczajnych (11.3) nie możemy, z powodów natury praktycznej, zastosować wprost lsode. Rzeczywiście, otrzymany układ równań zwyczajnych jest bardzo sztywny (im mniejsze  $h$ , tym sztywniejszy układ!), a więc powinniśmy rozwiązywać go, korzystając ze schematu niejawnego. Jednak w przypadku schematu niejawnego ujawnia się kluczowy niedostatek lsode: procedura DLSODE z biblioteki ODEPACK nie ma możliwości wykorzystania rozrzedzenia macierzy pochodnej i stosuje „w ciemno” *solver* dla macierzy gęstych, potężnie spowalniając działanie schematu niejawnego. Coś, co dotychczas było nieistotne (wcześniej rozważaliśmy układy co najwyżej kilku równań różniczkowych), w przypadku układu równań zwyczajnych (11.3), w którym mamy do czynienia z układem  $d$  równań różniczkowych zwyczajnych, gdzie

$$d = N_x \cdot N_y$$

(przy czym rozsądne wartości zarówno  $N_x$ , jak i  $N_y$  będą rzędu setek, a może nawet tysięcy), staje się ciężarem nie do udźwignięcia<sup>1</sup>.

Oczywiście, alternatywnym rozwiązaniem mogłoby potencjalnie być zastosowanie schematu jawnego, w którym w ogóle nie trzeba rozwiązywać żadnych równań, ani liniowych, ani tym bardziej nieliniowych! Jednak niestety, także i ten pomysł nie jest zbyt rozsądny. Jak się okazuje na gruncie teorii [15], musielibyśmy wtedy użyć bardzo małej długości kroku czasowego  $\tau$ , proporcjonalnego do  $\min\{h_x^2, h_y^2\}$ ! Dlatego, z wyjątkiem ekstremalnie krótkich czasów  $T$  lub niewielkiego  $d$  (maksymalnie rzędu kilkuset), schemat jawny po prostu „zaliczy się”, zmuszony do wykonania *bardzo wielu* kroków czasowych.

<sup>1</sup> Tak drastycznych ograniczeń nie mają funkcje MATLABa służące rozwiązywaniu sztywnych równań różniczkowych, np. `ode15s` — można (i trzeba!) wykorzystać tam rozrzedzenie macierzy pochodnej.

### 11.1. Rozwiązanie numeryczne równania zdyskretyzowanego w przestrzeni

Do rozwiązania nieliniowego parabolicznego równań różniczkowego cząstkowego (11.1), a dokładniej — (11.3) — wykorzystania biblioteki **CVODE**. Biblioteka CVODE z pakietu SUNDIALS [14] to kolekcja narzędzi bardzo dobrej jakości służących do rozwiązywania dużych układów równań różniczkowych zwyczajnych. W szczególności, można ją także dostosować właśnie do rozwiązywania ewolucyjnych równań różniczkowych cząstkowych metodą linii.

CVODE jest rozwinięciem biblioteki LSODE z pakietu ODEPACK, używanej w Octave do rozwiązywania równań różniczkowych. Jednak w odróżnieniu od LSODE, dostajemy możliwość rozwiązywania naprawdę dużych układów równań metodami niejawnymi, gdyż do zadań zlinearyzowanych, jakie trzeba rozwiązywać na każdym kroku czasowym, biblioteka CVODE może użyć nie tylko standardowej eliminacji Gaussa, ale także metody iteracyjnej (np. GMRES) — również z wykorzystaniem zadanej przez użytkownika macierzy ściiskającej (zob. wykład z [Matematyki Obliczeniowej II](#)). Drugie istotne rozszerzenie możliwości CVODE w stosunku do LSODE to możliwość pracy na komputerze wieloprocesorowym — lecz z niej nie będziemy tutaj korzystać.

Biblioteka CVODE jest bardzo dobrze udokumentowana, towarzyszy jej także zestaw przykładów wykorzystania jej możliwości. Właśnie najprostszym sposobem skorzystania z CVODE jest po prostu *ostrożne i uważne* zaadaptowanie przykładowego programu do własnych potrzeb.

Pierwszym krokiem powinno być ściągnięcie z internetu i zainstalowanie tej biblioteki w naszym systemie. W przypadku popularnych dystrybucji Linuxa będzie to bardzo proste, gdyż wystarczy zainstalować gotowy pakiet `sundials` (rzeczywiście, CVODE jest częścią większego pakietu, SUNDIALS). W przypadku bardziej egzotycznych systemów będziemy musieli samodzielnie skompilować kody źródłowe CVODE i następnie zainstalować w systemie. Na szczęście i ten proces jest szczegółowo opisany w podręczniku CVODE [13].

Poniżej pokazujemy zapis przykładowej instalacji biblioteki na oba sposoby.



Zobacz animację: *Instalacja biblioteki Cvode z pakietu RPM w systemie Fedora Linux*, znajdującą się na stronie WWW przedmiotu. Pokazujemy, jak na komputerze pracującym w systemie Fedora Linux zainstalować pakiet SUNDIALS, którego częścią jest biblioteka Cvode. Musimy pamiętać, by oprócz pakietu RPM z biblioteką zainstalować towarzyszący mu pakiet „devel”, zawierający m.in. pliki nagłówkowe potrzebne nam przy kompilacji własnych programów, korzystających z biblioteki.



Zobacz animację: *Instalacja biblioteki Cvode w systemie Linux, wykorzystująca kody źródłowe*. Procedura jest typowa dla instalacji innych bibliotek, których kod źródłowy pobraliśmy z internetu, znajdującą się na stronie WWW przedmiotu. Pokazujemy, jak na komputerze pracującym w systemie Fedora Linux zainstalować bibliotekę Cvode wprost z archiwum zawierającego kody źródłowe. Proces instalacji składa się z kilku etapów, na który składają się: ściągnięcie z internetu archiwum z plikami źródłowymi i dokumentacją, lektura instrukcji dotyczącej kompilacji biblioteki, modyfikacja `Makefile'a`, kompilacja, testy i na końcu — skopiowanie plików biblioteki do ich finalnego położenia.

Następnie, powinniśmy napisać program w języku C, który będzie rozwiązywał (11.3) z wykorzystaniem biblioteki CVODE. Typowy program wykorzystujący sekwencyjną wersję biblioteki CVODE ma mniej więcej następującą strukturę (por. podręcznik CVODE [13]):

```

/*****
pliki nagłówkowe, w tym ---
dla funkcji biblioteki CVODE
*****/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <cvode/cvode.h>
...itd...

/*****
definicje niezbędnych makr i stałych
*****/
#define T0 0.0 /* chwila początkowa */
#define TSTEP /* co jaki czas chcemy mieć rozwiązanie */
#define NSTEP /* ile takich rozwiązań? */
/* powyższe znaczy, że otrzymamy rozwiązania w chwilach  $T0 + i \cdot TSTEP$ ,  $i = 1..NSTEP$  */

#define RTOL RCONST(1.0e-5) /* parametr tolerancji błędu względnego */
#define ATOL RCONST(1.0e-6) /* parametr tolerancji błędu bezwzględnego */
#define NEQ (NX*NY) /* liczba równań w układzie, tutaj zadana jako  $NX \cdot NY$  */
...itd...

/*****
definicja struktury przechowującej dane,
jakie chcielibyśmy przekazywać do wewnątrz
solwera; przy rozsądnym wykorzystaniu zmiennych
globalnych i makrodefinicji, może być zbędna
*****/
typedef struct {
    realtype beta;
    ...inne obiekty...
} *UserData;

/*****
prototypy najważniejszych funkcji
definiowanych przez użytkownika:
- prawej strony (F) i opcjonalnie:
- Jakobianu (DFxV),
- być może także preconditionera;
ich format jest szczegółowo wyspecyfikowany
w manualu.
*****/
static int F(realtype t, N_Vector u, N_Vector udot, void *f_data);
static int DFxV(N_Vector V, N_Vector JV, realtype t,
    N_Vector y, N_Vector fy, void *jac_data, N_Vector tmp);

/*****
prototypy funkcji pomocniczych -
nazwy mówią same za siebie
*****/
static UserData AllocUserData(void);
static void InitUserData(UserData data);
static void FreeUserData(UserData data);
static void SetInitialProfiles(N_Vector u);

```

```

static int PrintOutput(void *ccode_mem, N_Vector u, realtype t, int iout);
static void PrintFinalStats(void *ccode_mem);
...itd...

/* ewentualne zmienne globalne dla (nie)wygody programowania */
...definicje zmiennych globalnych...

/*****
zaczyna się główny program
*****/
int main(void)
{
    /* KROK 0: niezbędne definicje zmiennych */

    realtype abstol = ATOL; /* tolerancja błędu */
    realtype reltol = RTOL;

    realtype t, tout; /* obsługa czasu */
    int iout;

    UserData data = NULL; /* kontener na parametry użytkownika */
    N_Vector U = NULL; /* tu znajdzie się rozwiązanie w chwili bieżącej */
    void *ccode_mem = NULL; /* będzie wskazywać obszar roboczy dla CVODE */

    /* KROK 1: alokacja i inicjalizacja danych użytkownika */

    U = N_VNew_Serial(NEQ);
    data = AllocUserData();
    InitUserData(data);
    SetInitialProfiles(U);

    /* KROK 2: przygotowanie CVODE do pracy:
    inicjalizacja obszarów roboczych, wybór parametrów solvera */

    /* wybieramy opcje pracy solvera: schemat BDF;
    równania nieliniowe będą rozwiązywane metodą Newtona */
    ccode_mem = CCodeCreate(CV_BDF, CV_NEWTON);
    CCodeSetFdata(ccode_mem, data);

    /* uwaga: to tu podajemy chwilę początkową T0! */
    CCodeMalloc(ccode_mem, f, T0, U, CV_SS, reltol, &abstol);

    /* wybieramy solver iteracyjny (GMRES, bez macierzy ścisłej)
    dla zadań zlinearyzowanych metodą Newtona */
    CVSpgrmr(ccode_mem, PREC_NONE, 20);

    /* opcjonalnie – podajemy funkcję wyznaczającą Jakobian */
    CVSpilsSetJacTimesVecFn(ccode_mem, DFxV, NULL);

    /* KROK 3: wydruk warunku początkowego – nie zaszkodzi! */

    iout = 0; tout = T0;
    PrintOutput(ccode_mem, U, tout, iout);

    /* KROK 4 (NAJWAŻNIEJSZY): rozwiązujemy równanie, a rozwiązania "drukujemy"! */

```

```

for (iout=1, tout = T0+TSTEP; iout <= NSTEP; iout++, tout += TSTEP)
{
    flag = CNode(cvode_mem, tout, U, &t, CV_NORMAL);
    PrintOutput(cvode_mem, U, t, iout);
    /* zawsze sprawdzamy, czy ostatni cykl obliczeń zakończył się sukcesem */
    if(check_flag(&flag, "CNode", 1)) break;
}

/* KROK 5: statystyki solvera i sprzątanie */

PrintFinalStats(cvode_mem);
N_VDestroy_Serial(U);
FreeUserData(data);
CNodeFree(&cvode_mem);

return(0);
}

```

Jak widzimy, nawet szkielet programu jest dość rozbudowany, a samo wywołanie biblioteki wymaga wielu przygotowań. Głównym zadaniem CVODE jest — mając zadaną funkcję prawej strony i dane początkowe — wyznaczyć wartość rozwiązania w danej chwili tout. Najważniejszą pętlą w powyższym kodzie jest więc

```

for (iout=1, tout = T0+TSTEP; iout <= NOUT; iout++, tout += TSTEP)
{
    flag = CNode(cvode_mem, tout, U, &t, CV_NORMAL);
    PrintOutput(cvode_mem, U, t, iout);
    if(check_flag(&flag, "CNode", 1)) break;
}

```

w której funkcja CNode() wyznacza wartości rozwiązania w chwilach tout, powiększanych za każdym obrotem pętli o TSTEP. Oczywiście, nic nie stoi na przeszkodzie, by zamiast pętli **for**, wprowadzonej w celu wygenerowania rozwiązań w równych odstępach czasu, skorzystać np. z pętli **while** i w niej decydować według własnych reguł o tym, w jakich momentach należy wydobyc wartość rozwiązania podobnie, jak robiliśmy to w Octave.

CVODE może oczywiście mieć kłopoty z dotarciem do zadanego czasu tout (na przykład dlatego, że rozwiązanie wcześniej eksploduje...) — dlatego zmienna t zawiera faktyczny czas, do jakiego udało się dotrzeć funkcji CNode(); jeśli kod zakończenia funkcji, który zapisujemy do zmiennej flag, jest różny od zera, oznacza to wystąpienie kłopotów w pracy CNode() i, w naszym przypadku, jest powodem do przerywania obliczeń.

Szczęśliwie, nasze zadanie — równanie (11.3) — jest bardzo podobne do przykładowego zadania towarzyszącego dokumentacji CVODE [13], cvkryx.c — dlatego będziemy je rozwiązywać poprzez modyfikację tego pliku.

```

/* ===== Modyfikacja pliku examples/cvode/serial/cvkryx.c ===== */
/* ===== Modyfikacja pliku examples/cvode/serial/cvDiurnal_kry.c ===== */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

#include <cvode/cvode.h> /* main integrator header file */
#include <cvode/cvode_spgmr.h> /* prototypes & constants for CVSPGMR solver */

```



To tylko fragment kodu źródłowego w C. Więcej na <http://mst.mimuw.edu.pl/lecture.php?lecture=ona&part=Ch11>.

Aby skompilować nasz plik do programu wykonywalnego, najwygodniej skorzystać z polecenia `make`, dla którego należy uprzednio przygotować plik z instrukcjami i opcjami kompilacji. Więcej o poleceniu `make` można przeczytać w [20] lub na przykład na [Ważniaku](#) (przedmiot *Środowisko programisty*). Przykładowy taki Makefile podajemy poniżej:

```
CC = gcc -std=c99
ARCHFLAGS= -march=native
CFLAGS = -O3 -msse2 -mfpmath=sse $(ARCHFLAGS) -funroll-loops \
→ -finline-functions -malign-double \
→ -ftree-vectorize -ftree-vectorizer-verbose=2
LIBS = -lm

all: cvrd

cvrd: cvrd.o
→ $(CC) $(CFLAGS) -o $@ $< -lsundials_cvode -lsundials_nvecserial -lm

%.o: %.c Makefile
→ $(CC) $(CFLAGS) -c $<

clean:
→ rm -f *.o
→ rm -f cvrd
```

Jak widzimy, w odróżnieniu do króciutkich plików Octave, gdzie na wyznaczenie rozwiązania (i jego wizualizację!) układu równań różniczkowych potrzebowaliśmy kilkunastu linii kodu, końcowa wersja pliku źródłowego w C jest bardzo obszerna. Dlatego ograniczymy się tutaj zwrócenia uwagi na kilka istotnych fragmentów głównego programu.

**Wykorzystanie makr** Zaczniemy od ogólnej uwagi, dotyczącej niebezpieczeństwa, jakie za sobą niesie używanie wyrażeń `#definiowanych`, takich jak np.

```
#define HX X/(N+1) /* krok przestrzenny */
```

Gdybyśmy napisali makro inaczej, `#define HX X/(N+1)` (różnica: brak nawiasów okalających), to potem spotkałaby nas niemiła niespodzianka:

```
#define HX X/(N+1) /* źle! */
z = a/HX;
```

Rzeczywiście, po podstawieniu makra do wyrażenia, dostajemy  $z = a/X/(N+1)$  i — ponieważ w języku C wartość wyrażenia oblicza się od lewej do prawej — zostałoby to obliczone jako  $(a/X)/(N+1) = a/(X \cdot (N+1))$ . To zapewne *nie jest* wynik, jakiego się spodziewalibyśmy....

**Dostęp do macierzy niewiadomych** Podobnie jak w Octave, dla danej chwili czasu, niewiadome traktujemy jako macierz  $U$  o rozmiarach  $N \times N$ . Jej elementy odpowiadają wartościom rozwiązania w punktach siatki przestrzennej. Ponieważ CVODE spodziewa się *wektora*, a nie macierzy niewiadomych — zastosujemy technikę rozwijania macierzy w wektor (por. [20]). Makro

```
#define LJth(a,i,j) (a[(j)-1 + ((i)-1)*N])
```

pozwole nam w funkcjach użytkownika (np. `F()` i `DFxV()`) patrzeć na  $U$  jak na macierz  $N \times N$  i jednocześnie pozwoli CVODE wewnętrznie traktować  $U$  jako wektor długości  $N \cdot N$ . Wszędzie, gdzie było to możliwe, pętle po wszystkich elementach macierzy  $U$  zapisujemy

```
for(i = 1; i <= N; i++)
{
    for( j = 1; j <= N; j++)
    {
        ...wyrażenie zawierające LJth(u,i,j)...
    }
}
```

Z punktu widzenia lokalności dostępu do danych, indeksowanie elementów macierzy  $U$  wierszami (właśnie tak, jak zapisano w makrze `LJth` powyżej) powinno być korzystniejsze niż analogiczne indeksowanie kolumnami. I rzeczywiście, pomiary czasu wykonania programu dla  $N = 320$  wykazują, że gdy wybierzemy indeksowanie wierszami, oszczędność czasu może sięgnąć kilkunastu procent. Jeśli zaś wybierzemy indeksowanie kolumnami i, bardziej złośliwie,  $N = 256$  (potęgę dwójki), wówczas spotyka nas spadek szybkości działania programu nawet o 40%. Jeszcze raz widzimy więc, jak ważne jest efektywne wykorzystanie pamięci podręcznej procesora (ang. *cache*).

**Opcje wykorzystania innych metod** W naszym kodzie zostawiliśmy sobie możliwość wykorzystania eliminacji Gaussa dla rozwiązywania układów równań z macierzą Jakobianu, w zależności od wartości parametru `ITLS`:

```
if (ITLS == 0)
    CVDense(cvode_mem, NEQ);
else
{
    CVSpgrmr(cvode_mem, PREC_NONE, 10);
    if(USERJAC > 0)
        CVSpilsSetJacTimesVecFn(cvode_mem, DFxV, NULL);
}
```

Dzięki temu, prowadząc próby dla niezbyt wielkich wartości  $N$ , zawsze możemy zbadać, czy ewentualne trudności *solvera* mają jakiś związek z kłopotami metody iteracyjnej. Dodatkowo, zwróćmy uwagę na opcję skorzystania — lub nie — z własnej procedury mnożenia przez Jakobian. Często bowiem opcja automatycznego wyznaczenia wyniku na podstawie różnic dzielonych jest w ostatecznym rachunku tańsza, zwłaszcza, gdy nieliniowości nie są dokuczliwe. Poza tym zawsze warto upewnić się, że faktycznie wykorzystanie funkcji `DFxV()` powoduje zmniejszenie, a nie zwiększenie liczby iteracji Newtona w stosunku do wariantu z automatyczną aproksymacją Jakobianu (pogorszenie zbieżności metody Newtona jest najczęściej objawem naszego błędu w definicji `DFxV()`).

**Zapis wyników do pliku i ich wizualizacja** Wreszcie, rozwiązując równanie różniczkowe, musimy mieć świadomość, że wynikiem będzie mrowie liczb. Sensowne jest więc pomyśleć o ich zapisie do pliku, a następnie — wizualizacji.

Nasza procedura jest tak skonstruowana, by zapisany plik z danymi bez trudu można było wczytać do zewnętrznego programu wizualizacyjnego [OpenDX](#):

```

outfile = fopen("cvrd.out", "w");
/* ...pominięte instrukcje... */

for(i = 1; i <= N; i++)
{
    for( j = 1; j <= N; j++)
    {
        fprintf(outfile, "%e", IJth(udata,i,j));
    }
    fprintf(outfile, "\n");
}
/* ...pominięte instrukcje... */
fclose(outfile);

```

Dane z macierzy, odpowiadające ich ułożeniu na siatce dwuwymiarowej, zapisujemy dla kolejnych kroków czasowych. Gdyby danych było bardzo dużo, możemy je skompresować, zapisując je w postaci binarnej, także akceptowanej przez OpenDX.

Aby zapisać dane w formacie strawnym dla innego popularnego systemu, [Paraview](#), należy użyć nieco innego kodu, powodującego utworzenie *serii* plików zawierających „zdjęcie” układu dla danej chwili czasowej, indeksowanej w naszym programie, jak pamiętamy, zmienną iout. Na każdym kroku czasowym, macierz wartości rozwiązania w punktach siatki przestrzennej,  $U_{ij} \leftrightarrow u(t_k, x_i, y_j)$  musimy zapisać w specjalnym formacie. W tym celu możemy użyć następującego kodu, który zamieścimy wewnątrz funkcji PrintOutput() programu cvrd.c:

```

{
char filename[26];

sprintf(filename, "cvrd%d.vtk", iout);
outfile = fopen(filename, "w");

fprintf(outfile, "#_vtk_DataFile_Version_3.0\n");
fprintf(outfile, "CVRD_wyniki_dla_siatki_%d_x_%d_Krok_czasowy:%d/%d\n", N, N, iout, NOUT);
fprintf(outfile, "ASCII\n");

fprintf(outfile, "DATASET_STRUCTURED_POINTS\n");
fprintf(outfile, "DIMENSIONS_%d_%d_%d\n", N, N, 1);
fprintf(outfile, "ORIGIN_%g_%g_%g\n", 0.0, 0.0, 0.0);
fprintf(outfile, "SPACING_%g_%g_%g\n", HX, HY, 0.0);

fprintf(outfile, "POINT_DATA_%d\n", N*N);
fprintf(outfile, "SCALARS_U_float\n");
fprintf(outfile, "LOOKUP_TABLE_default\n");
for( j = 1; j <= N; j++)
{
    for(i = 1; i <= N; i++)
    {
        fprintf(outfile, "%e\n", IJth(udata,i,j));
    }
}

fclose(outfile);
}

```



Nie będziemy tu dyskutować, dlaczego właśnie warto użyć takiej, a nie innej reprezentacji danych dla ich późniejszej wizualizacji w systemie OpenDX albo ParaView. Zainteresowanych odsyłamy do dokumentacji tych środowisk lub do podręcznika [20].

W zamian, w rozdziale 11.2 przyjrzymy się (dosłownie, śledząc zapis działań na komputerze), jak wizualizować tak spreparowane dane.

## 11.2. Wizualizacja wyników

Poniżej przedstawiamy zapis sesji wizualizacyjnej, w której dane wygenerowane podczas symulacji wczytamy do dwóch narzędzi wizualizacyjnych: OpenDX i ParaView. Ich instalacja nie powinna nastęczać żadnych problemów, pod warunkiem, że używana przez nas dystrybucja Linuxa dostarcza gotowe pakiety instalacyjne (por. demo instalacji CVODE z pakietów RPM w Fedorze wcześniej w tym rozdziale); kompilacja tych pakietów z plików źródłowych może być źródłem frustracji i poważnego bólu głowy.

### 11.2.1. Wizualizacja w OpenDX



Zobacz animację: *Demonstracja sposobu wizualizacji wygenerowanego pola w systemie OpenDX*, znajdującą się na stronie WWW przedmiotu. Pokazujemy, jak wygenerować dane do wizualizacji, opisać je korzystając z Data Promptera, a następnie wizualizować. Potem demonstrujemy najprostsze sposoby manipulacji obrazem: animację, powiększanie i obracanie.

### 11.2.2. Wizualizacja w ParaView



Zobacz animację: *Wizualizacja danych z symulacji równania reakcji-dyfuzji w ParaView*, znajdującą się na stronie WWW przedmiotu. Pokazano m.in., jak uruchamiać Paraview, wczytać zestaw danych, nałożyć proste filtry, takie jak Extract Surface, Wrap By Scalar, Threshold i Slice, interpretować wyniki, zapisać wynik wizualizacji do pliku PNG i kończyć pracę.

Korzystając z dodatkowych możliwości ParaView, możemy zwizualizować nie tylko samo pole, ale także na przykład jego gradient i dywergencję.



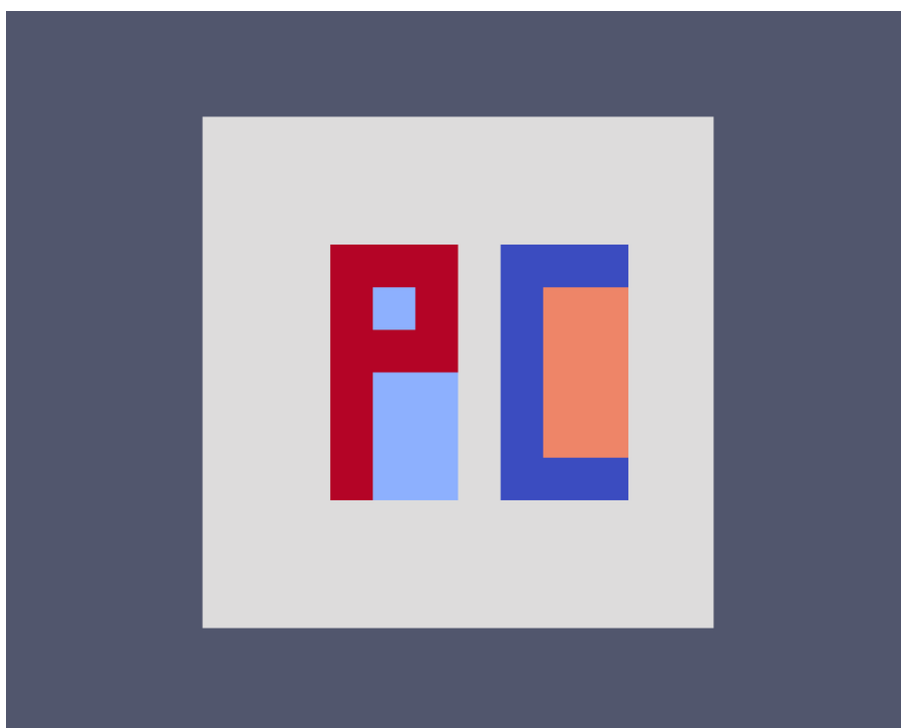
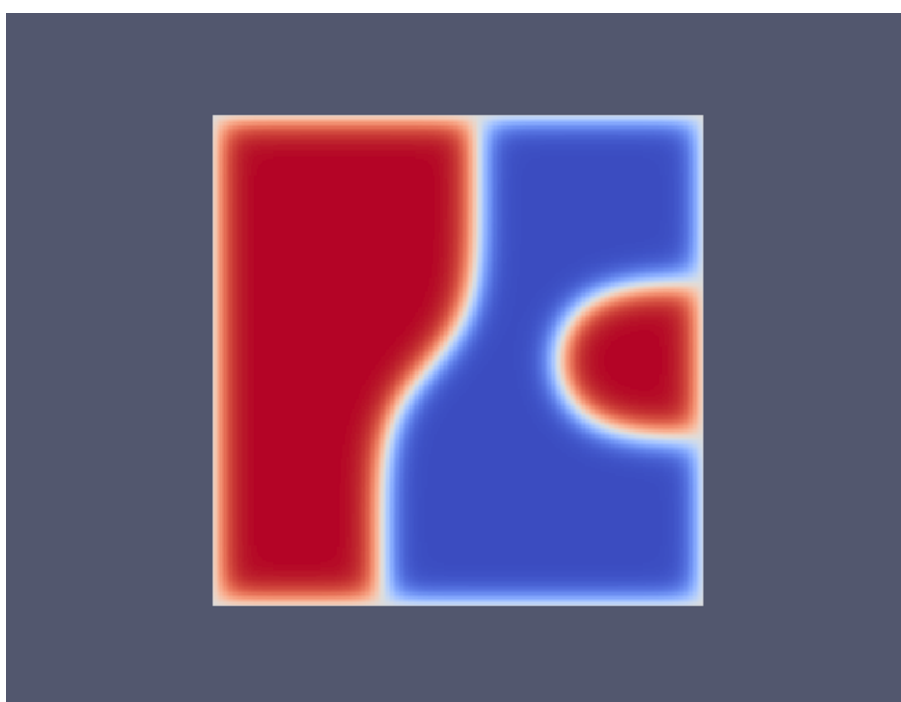
Zobacz animację: *Wizualizacja gradientu i dywergencji pola z symulacji równania reakcji-dyfuzji w ParaView*, znajdującą się na stronie WWW przedmiotu. Gradient wizualizujemy stosując gotowy filtr, dywergencję musimy sami skonstruować, korzystając z filtra Calculator.

## 11.3. Weryfikacja wyników

Na zakończenie warto, abyśmy wstępnie zeweryfikowali uzyskane wyniki. Do testów użyliśmy warunku początkowego przedstawionego na rysunku 11.2.

Korzystając z CVODE i wizualizując rozwiązanie w ParaView, dostaliśmy w chwili  $T = 150$  rozwiązanie przedstawione na rysunku 11.3.

Oczywiście po wcześniejszych naszych doświadczeniach nasuwa się pytanie, ile pewności możemy mieć co do jakości otrzymanych rozwiązań? Ponieważ nie mamy czasu ani siły na opracowanie bardziej wyrafinowanego testu, możemy skorzystać ze standardowego *tricku*: należy

Rysunek 11.2. Rozwiązanie w chwili  $T = 0$ .Rysunek 11.3. Rozwiązanie w chwili  $T = 150$ , siatka o  $N_x = N_y = 256$  węzłach,  $\text{RTOL}=1\text{e}-5$ .

dwukrotnie (lub więcej razy!) zmniejszyć wszystkie parametry dyskretyzacji i tolerancji błędów — i sprawdzić, czy dalej dostajemy „te same” wyniki.

W naszym przypadku:

- czterokrotnie zmniejszymy krok dyskretyzacji przestrzennej, czyli zmienimy parametr  $N$  z 256 na 1024;
  - stukrotnie zmniejszymy parametry tolerancji błędów po czasie, czyli zmienimy  $RTOL$  z  $1e-5$  na  $1e-7$ ;
- i ponownie przeprowadzimy symulację i wizualizację wyników. Jak widać<sup>2</sup> z rysunków 11.3 oraz 11.4, uzyskane wyniki są *bardzo podobne*, a więc (*prawdopodobnie!*) sensownie przybliżają one prawdziwe rozwiązanie.



Rysunek 11.4. Rozwiązanie w chwili  $T = 150$ , siatka o  $N_x = N_y = 1024$  węzłach,  $RTOL=1e-7$ .

**Ćwiczenie 11.1.** Zapisywanie wyników na potrzeby wizualizacji w formie liczb podwójnej precyzji zazwyczaj jest grubą przesadą. Zmodyfikuj więc procedurę eksportu rozwiązania do pliku w taki sposób, by zapisywać wartości jedynie w pojedynczej precyzji (**float**) i tym samym dwukrotnie zmniejszyć objętość plików.

**Ćwiczenie 11.2.** Rozwiąż równanie Swifta–Hohenberga, modelujące powstawanie wzorzystych struktur z początkowego chaosu,

$$\phi_t = r\phi - (\Delta + 1)^2\phi + \phi^2 - \phi^3 \quad \text{w } \Omega \times (0, T].$$

Eksperymentuj z różnymi obszarami  $\Omega$  (na początek: z kwadratem), różnymi nieujemnymi (w tym: zerowymi) wartościami parametru  $r$  i różnymi warunkami brzegowymi dla  $\phi$ . Startując z losowego rozkładu  $\phi(x, 0)$  odpowiedz na pytanie, jaka jest wartość uzyskanych wyników. (Trudne.)

<sup>2</sup> Znacznie korzystniej jest *zmierzyć* różnicę pomiędzy wynikami. Oko ludzkie może nie dostrzec niewielkich, lokalnie występujących, ale systematycznie utrzymujących się różnic, będących symptomem kłopotów *solvera*.

## 12. Czy te symulacje mogą kłamać?

Symulacje komputerowe na dobre zadamowiły się w wielu dziedzinach badań naukowych, a tym bardziej w zastosowaniach modelowania matematycznego. Ufamy — ludzie matematyki stosowanej — wynikom symulacji tak bardzo, że przestaliśmy głośno wątpić w ich poprawność. Doszło do tego, że w pracach naukowych dotyczących analizy konkretnych modeli matematycznych rzeczywistości nagminnie pomija się szczegóły dokonanych obliczeń (zastępując je — jeśli w ogóle — zdawkowym „...wyniki komputerowych symulacji wykazują, że...”).

Gdy porównamy to ze szczegółowym opisem narzędzi badawczych, technik pomiarowych i procedury eksperymentów, który jest nieodłączną częścią poważnych prac naukowych, będziemy mogli lepiej pojąć stopień bałwochwalczego uwielbienia i bezgranicznej ufności, jaką współcześni badacze obdarzają narzędzia i procedury prowadzenia *eksperymentów obliczeniowych*.



Zobacz ilustrację: *Występujący w Oceanie Atlantyckim stawonóg, o nazwie skrzypłocz, znajdującą się na stronie WWW przedmiotu. Ze względu na wygląd, skrzypłocze po angielsku nazywa się horeseshoe crab, czyli krab-podkowa. Należą do rzędu ostrogonów, z powodu odwłoka w kształcie długiego kolca (miecza)*

Aby w życiu zawodowym uniknąć takiego — zdecydowanie niewłaściwego — stosunku do obliczeń naukowych, proponuję na zakończenie „zbadać” metodami komputerowymi kolejny model matematyczny: tym razem będzie to model reakcji oka [skrzypłocza](#) na światło, zaczerpnięty z pracy [6]<sup>1</sup>. Jest to układ  $N$  równań różniczkowych na  $[0, T]$  postaci

$$y_r' + y_r(1 + y_N) = y_{r-1}, \quad r = 1, \dots, N. \quad (12.1)$$

(Jak zwykle w modelowaniu matematycznym, powyższe eleganckie równanie jest efektem wcześniejszych sprytnych manipulacji wzorami i dewymiarowania oryginalnego układu równań). Zadana z góry funkcja  $y_0(t)$  ma modelować przebieg natężenia światła zewnętrznego w czasie. W [6] jest to funkcja schodkowa, przyjmujemy więc konkretnie, że dla  $t \in [0, T]$  i zadanych czasów krytycznych  $T_1 < \dots < T_N \in [0, T]$

$$y_0(t) = \begin{cases} 1, & \text{gdy } \min_i |t - T_i| < \tau, \\ 0 & \text{w p.p.} \end{cases}$$

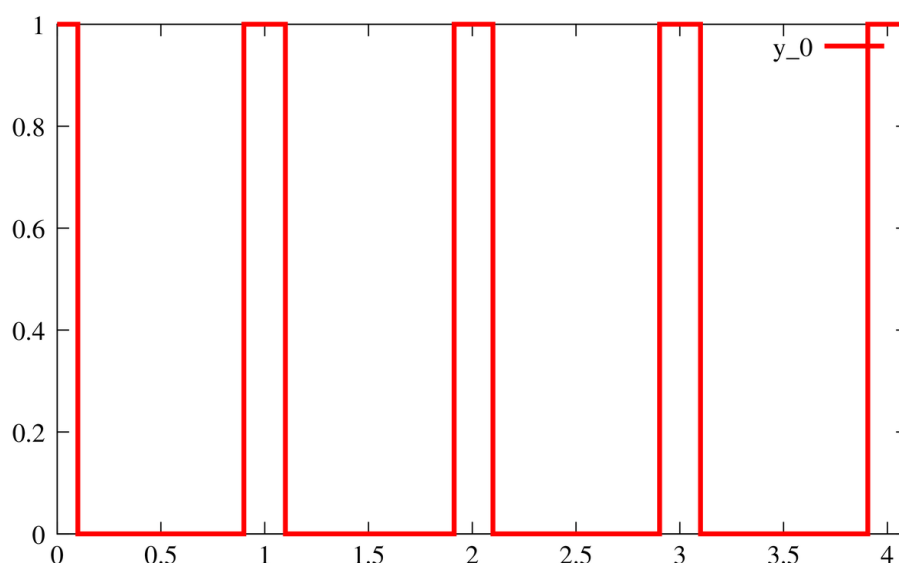
(por. rysunek 12.1).

Dodatkowo powinniśmy przyjąć, że  $\tau$  jest małe. Dla badaczy interesująca jest przede wszystkim składowa  $y_N(t)$  rozwiązania, modelująca odpowiedź oka na bodziec.

Oryginalna praca [6] była na tyle stara, by — oprócz szczegółów sposobu przeprowadzania pomiarów fizycznych, służących jako porównawczy materiał eksperymentalny — wskazać użyte narzędzia obliczeniowe; oto one [6, strona 257]:

Solutions [...] were obtained on EDSAC 2 at Cambridge using a Runge–Kutta method.

<sup>1</sup> Dziękuję p. Wandzie Niemyskiej za przedstawienie modelu i wskazanie literatury.



Rysunek 12.1. Przykładowy wykres funkcji  $y_0(t)$ , gdy  $T_i = 0, 1, 2, \dots$  i  $\tau = 0.1$ .

Każdy, kto choć trochę liźnął metod numerycznych rozwiązywania równań różniczkowych zwyczajnych, natychmiast będzie miał wątpliwości: *którą konkretnie* metodę Rungego–Kutty mają na myśli autorzy? czy klasyczną metodę czteropoziomową, czy może tańszą, dwupoziomową? A może którąś z metod RKF (o ile mamy w ogóle świadomość istnienia takiego wariantu metody Rungego–Kutty...)? Czy stosowano automatyczne sterowanie długością kroku — jeśli tak, to na podstawie jakiego kryterium? A jeśli nie, to jakiej użyto długości kroku całkowania?

## 12.1. Pierwsze błędne symulacje

Na tym etapie nie wydaje się, by powyższe równanie mogło sprawić nam (to znaczy: naszemu systemowi obliczeń numerycznych) jakikolwiek poważniejszy problem. Wystarczy skorzystać z gotowych narzędzi rozwiązywania równań różniczkowych zwyczajnych dostępnych w Octave (lub w MATLABie): na przykład, z *solvera* **ode45** zaimplementowanego w obu wspomnianych systemach obliczeń naukowych.

W naszych obliczeniach przyjmiemy na początek następujące wartości liczbowe parametrów zadania:

Parametr	Wartość
$N$	10
$T$	3
$\tau$	0.2

Rozwiązanie problemu i wizualizacja składowej  $y_N$  mogłaby być na przykład zrealizowana poniższym, całkiem standardowym kodem:

```
function dY = rhs(t,Y)
% prawa strona rownania
N = length(Y);
q = [y0(t); Y(1:N-1)];
dY = -Y*(1+Y(N)) + q;
end
```

```
function v = y0(t)
% wymuszenie zewnętrzne; t musi być skalarem!
global tau;
global Ti;
v = (min(abs(t - Ti)) < tau); % krótkie (2τ) pulsy wokół Ti
end
```

```
global tau;
global Ti;

tau = 0.2;
N = 10;
T = 3;
Ti = 0:T; % Ti = 0, 1, ..., [T]

Y0 = zeros(N,1);
tsmpl = linspace(0,T,300);

[tcomp, Y] = ode45(@rhs, tsmpl, Y0);
plot(tcomp, Y(:,N));
```

Program działa w Octave i wydaje się, że produkuje poprawne wyniki ( $y_N$  stabilizuje się), ale już dla  $T = 153$  obserwujemy ciekawe zjawisko: wyznaczone rozwiązanie wykazuje zachowanie oscylacyjne!

**Ćwiczenie 12.1.** W symulacji powyżej połącz  $T = 153$  i sprawdź wyniki. Jak wygląda przebieg wszystkich składowych rozwiązania w czasie?

A więc mamy „numeryczny dowód”, że oczy skorupiaka (przynajmniej w ramach naszego modelu), poddane pulsacyjnemu działaniu światła zewnętrznego, reagują także w sposób periodyczny — co zasadniczo nie powinno nas dziwić... Warto więc byłoby zbadać okres tej reakcji, a następnie spróbować odnieść to zachowanie do wyników eksperymentów fizycznych. Jednak... fizyczny eksperyment nie może być przeprowadzony w tak długim czasie, pozostaje więc nam poruszać się na gruncie teorii.

**To, że program działa, nie znaczy jeszcze, że działa tak, jak byśmy chcieli** Gdybyśmy byli nieco bardziej ostrożni, zauważylibyśmy (być może), iż nieco *nadużyliśmy* składni funkcji **ode45**. Dokładniej, aby skorzystać z automatycznego sterowania długością kroku w tym *solverze*, powinniśmy<sup>2</sup> — nieco inaczej niż w *lsode* — podać jako drugi argument jedynie dwuelementowy wektor  $[0, T]$ ! W przeciwnym razie, *solver* zawsze wybiera stały krok długości  $\text{tsmpl}(i) - \text{tsmpl}(i-1)$ , co przy  $T \approx 150$  daje krok całkowania  $\approx 0.5$  — zdecydowanie za duży, na przykład, w porównaniu z  $\tau \approx 0.2$ !

**Ćwiczenie 12.2.** Sprawdź, jak zmieniają się rozwiązania, gdy układ równań różniczkowych (12.1) zastąpimy układem postaci [6]

$$y'_r + y_r e^{y_N} = y_{r-1}, \quad r = 1, \dots, N.$$

Przeprowadź symulacje i zweryfikuj ich wyniki. A może da się coś konkretnego powiedzieć na gruncie matematycznej teorii, korzystając z faktu, że (w jakimś sensie)  $e^y \approx 1 + y$ ?

<sup>2</sup> Wystarczy przeczytać opis funkcji!

## 12.2. Kłopotliwe pytania

Po rozwiązaniu zagadki oscylacji — nieistniejących w rzeczywistości — możemy teraz mieć ochotę na zabawę z wartością parametru  $\tau$ , określającego długość trwania impulsu świetlnego. Z punktu widzenia modelowania, interesujące byłoby wyznaczenie wartości  $\tau$ , przy której bezwładność oczu stwora jest na tyle duża, że przestają one reagować na (najwyraźniej zbyt krótkie) impulsy świetlne i ich reakcja wygasa.

W tym celu wystarczy stopniowo zmniejszać  $\tau$  i obserwować, jak zmieni się odpowiedź układu; dla wystarczająco krótkich impulsów spodziewamy się, że bodźce będą zbyt słabe, by stabilnie wzbudzić niezerową wartość składowej  $y_N$ : spodziewamy się  $y_N$  gasnących do zera.

Przeprowadźmy zatem symulację dla  $\tau = 10^{-2}$  — to znaczy dwudziestokrotnie mniejszej niż dotychczas.

```
global tau;
global Ti;

tau = 1e-2; % krotsze impulsy
N = 10;
T = 3; % zbyt krótki czas, by cos zobaczyc
Ti = 0:T;

Y0 = zeros(N,1);
tsmpl = linspace(0,T,300);

[tcomp, Y] = ode45(@rhs, [0,T], Y0); % juz dobra informacja o przedziale calkowania
plot(tcomp,Y(:,N));
```

Jak można zorientować się z przeprowadzonej symulacji, początkowo  $y_N$  rośnie (eksponencjalnie?) i nie wiadomo, czy potem stabilizuje się, jak poprzednio, czy też może zachowanie jest inne. Dlatego należy wydłużyć czas symulacji:

```
global tau;
global Ti;

tau = 1e-2; % krotsze impulsy
N = 10;
T = 30; % dluzszy czas symulacji
Ti = 0:T;

Y0 = zeros(N,1);
tsmpl = linspace(0,T,300);

[tcomp, Y] = ode45(@rhs, [0,T], Y0); % juz dobra informacja o przedziale calkowania
plot(tcomp,Y(:,N));
```

Okazuje się, że faktycznie, rozwiązania  $y_N$  po chwilowym wzbudzeniu szybko gasną do zera. Aby upewnić się, czy nie mamy czasami do czynienia z numerycznym artefaktem, rozwiążemy to samo równanie, drastycznie zaostrzając kryteria tolerancji błędów, używając do tego funkcji `odeset`.

```
global tau;
global Ti;

tau = 1e-2;
N = 10;
```

```

T = 30;
Ti = 0:T;

Y0 = zeros(N,1);
tsmpl = linspace(0,T,300);

[tcomp, Y] = ode45(@rhs, [0,T], Y0);
[tcomps, Ys] = ode45(@rhs, [0,T], Y0, odeset('RelTol',1e-9,'AbsTol', 1e-9)); % zmiana parametrów pracy solvera; domyslnie
plot(tcomp,Y(:,N), tcomps,Ys(:,N));

```

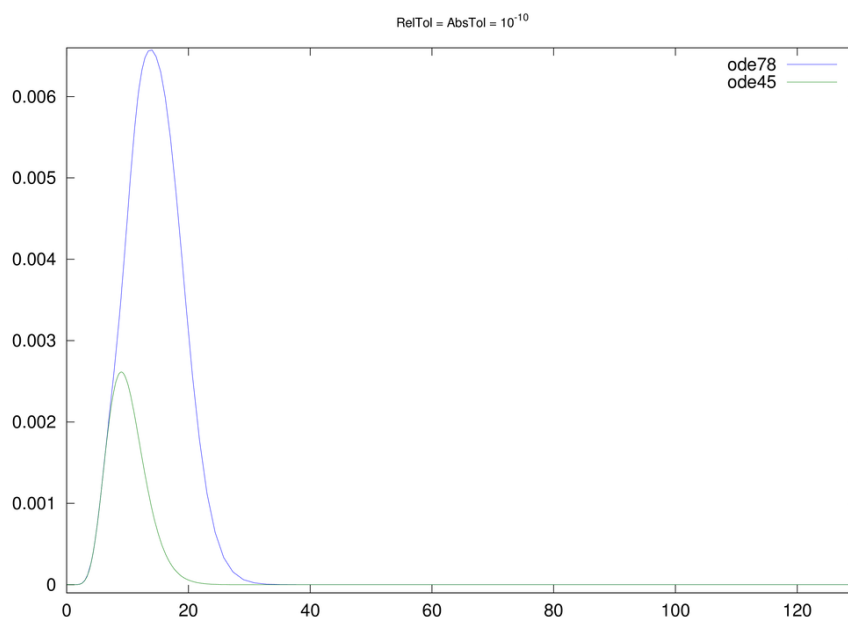
Jak widać, choć otrzymane krzywe nie pokrywają się idealnie, zgadzają się ze sobą co do charakteru:  $y_N$ , po chwilowym wzroście, gaśnie do zera.

Gdybyśmy jednak wykazali mniej zaufania do wyników i rozwiązali zadanie... *innym solverem*, na przykład ode78 (dostępny w Octave, w MATLABie możesz wybrać na przykład ode115)?

**Ćwiczenie 12.3.** Wykonaj powyższą symulację z innym *solverem*.

*Wskazówka.* Jeśli będziesz pracować z *Isode*, pamiętaj o tym, że wymaga on innej składni wywołania oraz innej składni funkcji prawej strony!

Może okazać się, że otrzymamy zupełnie inny wykres! Co prawda pragmatycy staraliby się podkreślić, że „co do charakteru” dalej mamy zgodność, bo rozwiązania gasną (do zera?), ale większość z nas, widząc wykresy jak na rysunku 12.2, zaczęłaby powątpiewać w jakość prowadzonych obliczeń.



Rysunek 12.2. Porównanie wykresów  $y_N(t)$  dla  $\tau = 10^{-2}$ , wyznaczonych różnymi solverami, dla  $T = 130$ .

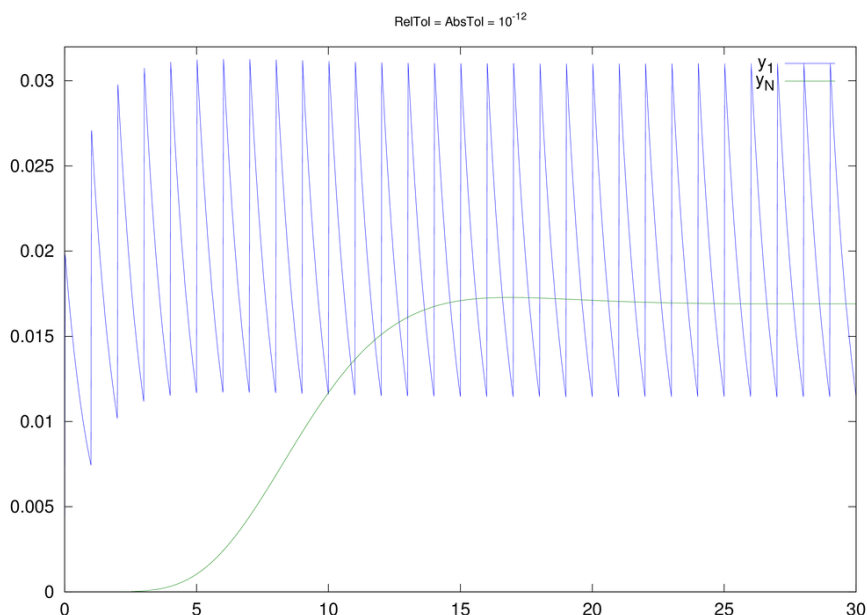
Może jednak problem leży w tym, że impulsy są na tyle krótkie, że nasz *solver*, stosując pewne strategie doboru kroku całkowania „opuszcza” część impulsu tak, że to z tego powodu rozwiązania gasną?

Gdyby bowiem zastosować *jeszcze bardziej drastyczne* parametry tolerancji błędów, na przykład

```
odeset('RelTol',1e-12,'AbsTol', 1e-12)
```



to okazuje się, że co prawda czas obliczeń wydłuża się niepokojąco, ale też i wynik obliczeń zmienia się nie do poznania, zob. wykres 12.3.



Rysunek 12.3. Wykres  $y_1(t)$  i  $y_N(t)$  dla  $\tau = 10^{-2}$ , wyznaczony przy bardziej wyśrubowanych parametrach tolerancji błędu, solverem ode45..

Po doświadczeniu z większymi  $\tau$  bylibyśmy zapewne skłonni uwierzyć bez zastrzeżeń, że otrzymaliśmy wynik wykazujący, że dla  $\tau = 10^{-2}$  rozwiązania jednak nie zanikają w czasie.

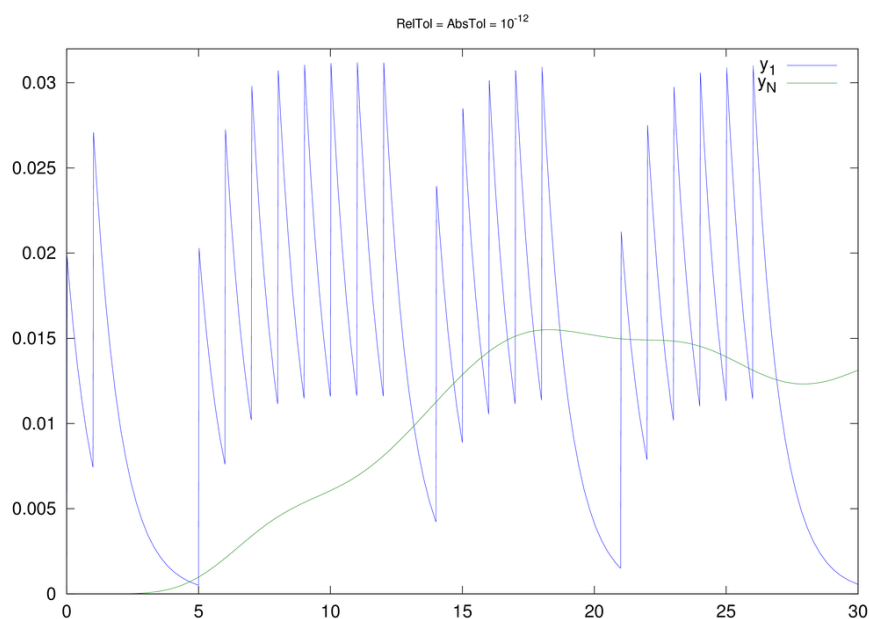
Ale, jeśli przeprowadzilibyśmy symulację korzystając z *dokładniejszej* metody, ode78 i równie ostrych parametrach tolerancji błędu jak ostatnio, to zapewne zaskoczyłby nas przebieg tak wyznaczonego rozwiązania, przedstawiony na rysunku 12.4.

Bardzo możliwe, że jednak za bardzo przejęliśmy się narzędziami, a za mało samym problemem: przecież rozwiązania, które nas interesują, są bardzo mało regularne! A to z kolei znaczy, że stosowanie *solwera* wysokiego rzędu aproksymacji — takiego jak ode78 — nie bardzo ma sens; rzeczywiście, dla parametrów tolerancji błędu  $10^{-14}$ , wyniki symulacji (prawie) pokrywają się (por. rysunki 12.5 i 12.6: nie tylko dla obu dotychczas przez nas próbowanych *solwerów*, ale także dla lsode, który wykorzystuje nie tylko zmiany długości kroku, ale także dostosowuje rząd metody).

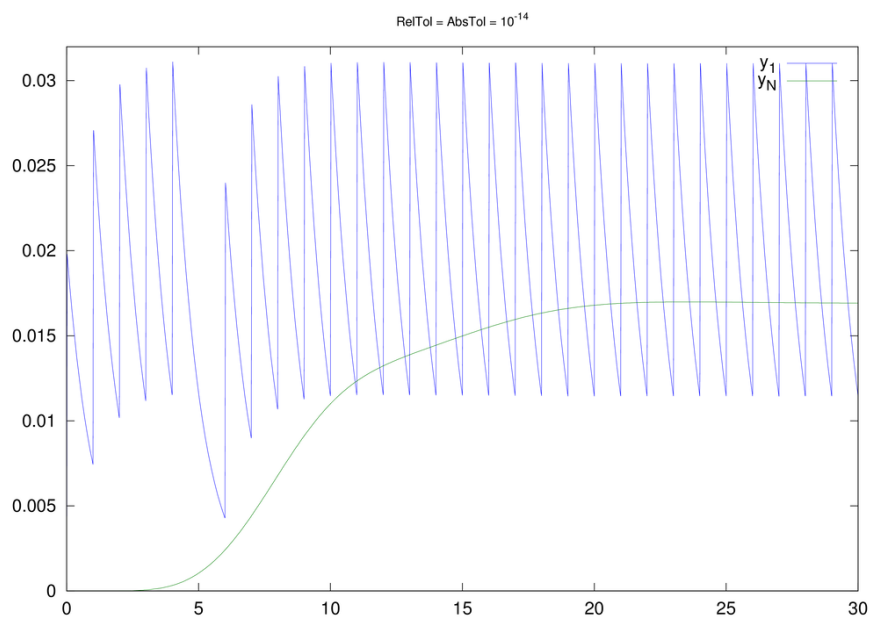
A jak *Ty* uważasz, poniżej jakiej wartości  $\tau$  rozwiązania  $y_N$  *naprawdę* dążą do zera, gdy  $t \rightarrow \infty$ ?

### 12.3. Ponure konstatacje

Zauważmy, że nasze zadanie jest mimo wszystko *trudne* do rozwiązania numerycznego! Wiąże się to z charakterem nieliniowości prawej strony równania, a dokładniej, z przebiegiem  $y_0(t)$ , która jest funkcją nieciągłą. Wszystkie używane przez nas metody (w postaci *solwerów* typu **ode45**) zakładają tymczasem, że aproksymowane rozwiązania są funkcjami *dostatecznie gładkimi*, co oczywiście w naszym przypadku nie jest założeniem spełnionym. To typowa dla obliczeń naukowych sytuacja, gdy istniejące oprogramowanie (i zaimplementowane w nim metody numeryczne) zmuszamy do pracy w warunkach wykraczających poza ich założone zastosowanie.



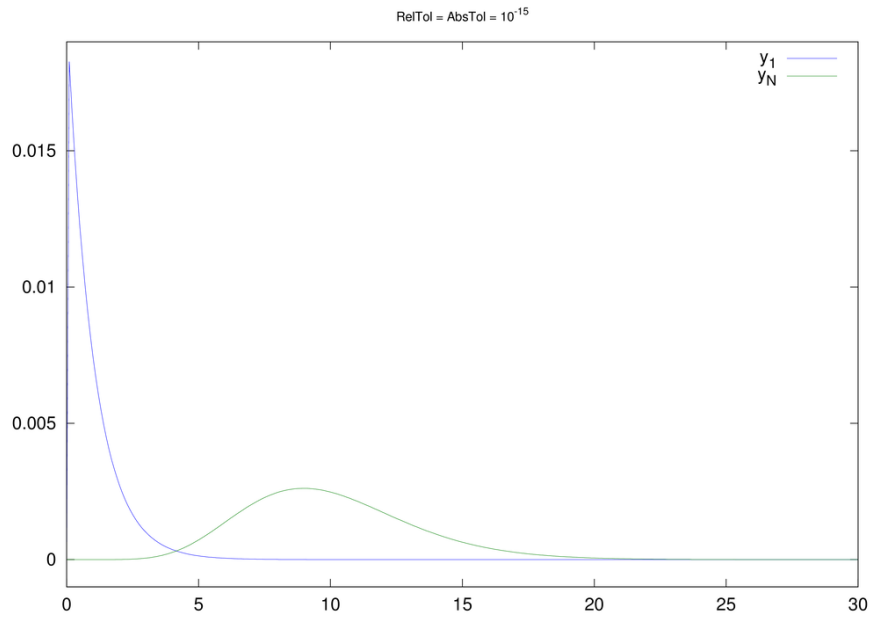
Rysunek 12.4. Wykres  $y_1(t)$  i  $y_N(t)$  dla  $\tau = 10^{-2}$ , wyznaczony przy bardziej wyśrubowanych parametrach tolerancji błędu, przez solver wyższego rzędu..



Rysunek 12.5. Wykres  $y_N(t)$  dla  $\tau = 10^{-2}$ , wyznaczony przy jeszcze bardziej wyśrubowanych parametrach tolerancji błędu, przez solver wyższego rzędu, ode78..

Gdyby na nasz problem rzucić okiem praktyka, natychmiast zwróciłby nam uwagę na to, że w „trudnej” wersji zadania (dla małych  $\tau$ ) występują dwie skale czasowe: skala szybkości, związana z zewnętrznymi impulsami i skala wolności, reakcji  $y_N$  na wymuszenie. To zaś podpowiada nam, by w takiej sytuacji zastosować krok całkowania *krótszy* niż długość najkrótszej skali czasowej zadania.

Rozwiązaniem siłowym byłoby więc *zmuszenie solvera* do pracy z krokiem czasowym wyraż-



Rysunek 12.6. Wykres  $y_N(t)$  dla  $\tau = 10^{-2}$ , wyznaczony przy średnio wyśrubowanych parametrach tolerancji błęd, przez inny solver, zmiennego rzędu, lsode..

nie mniejszym niż  $\tau$ . Jednak z drugiej strony oznacza to nieprawdopodobne zwiększenie *kosztu* obliczeniowego potrzebnego do wyznaczenia rozwiązania: dla  $T = 10$  i  $\tau = 10^{-3}$  musielibyśmy wykonać *co najmniej*  $10^5$  kroków czasowych (zapewne znacznie więcej).

I faktycznie, o ile uzyskanie (najprawdopodobniej) błędnego wykresu  $y_N$  wymagało tylko kilku-kilkunastu sekund cierpliwości, to wygenerowanie (bliższego prawdy, miejmy nadzieję) wykresu  $y_N$  kodem:

```
opts = odeset('RelTol', 1e-10, ...
    'AbsTol', 1e-10, ...
    'MaxStep', tau, ... % redukcja kroku całkowania do najmniejszej skali czasowej
    'InitialStep', tau/3);
[tcomp, Y] = ode45(@rhs, [0,T], Y0, opts);
```

ciągnie się niemalże w nieskończoność (sprawdź, ile herbat zdążysz sobie przyrządzić, gdy  $\tau = 10^{-4}$ ...).

Czy można byłoby więc jakoś temu zapobiec?

## 12.4. Światelko w tunelu

Musimy raz jeszcze spojrzeć na *naturę* naszych kłopotów. Wszystko zdaje się pochodzić stąd, że *solvery*, których używamy, dobrze działają, gdy rozwiązania są gładkie. Tymczasem nasze rozwiązania są niegładkie. No dobrze, ale przecież tym samym jest to nie tylko problem obliczeniowy, ale także problem czysto matematyczny: jak bowiem zdefiniować rozwiązania równania różniczkowego, w którym prawa strona jest... nieciągła?!

W naszym przypadku, ściślej rzecz biorąc, prawa strona jest kawałkami stała — ciągła z wyjątkiem punktów postaci  $T_i \pm \tau$  — zatem, gdyby ograniczyć się do przedziałów postaci

$$\underbrace{(T_i - \tau, T_i + \tau)}_{v_0=1} \quad \text{oraz} \quad \underbrace{(T_i + \tau, T_{i+1} - \tau)}_{v_0=0},$$

tam moglibyśmy sensownie zdefiniować nasze równanie różniczkowe (za wartość początkową w danym przedziale biorąc wartość końcową z przedziału poprzedniego). Gdybyśmy więc postąpili identycznie w naszych obliczeniach, całkowicie uniknęlibyśmy w ten sposób kłopotów z nieciągłościami  $v_0$  i ich automatyczną lokalizacją! Ceną za to jest wbudowanie w program pętli po pododcinkach, jednak — jak sami za chwilę się przekonamy — będzie warto ją zapłacić!

```

opts = odeset('RelTol', RTOL, ...
    'AbsTol', ATOL, ...
    'InitialStep', tau/3);
tcomp = []; Y = []; Y0i = Y0;
for i = 1:length(Ti)-1
    opts = odeset(opts, 'MaxStep', (Ti(i+1)-Ti(i))*0.1);
    [tcomp, Yi] = ode45(@rhs, [Ti(i),Ti(i+1)-tau], Y0i, opts);
    tcomp = [tcomp; tcomp];
    Y = [Y; Yi];
    Y0i = Yi(end,:);

    opts = odeset(opts, 'MaxStep', tau*0.1);
    [tcomp, Yi] = ode45(@rhs, [Ti(i+1)-tau,Ti(i+1)], Y0i, opts);
    tcomp = [tcomp; tcomp];
    Y = [Y; Yi];
    Y0i = Yi(end,:);
end
I = Y(:,1); V = Y(:,N);
plot(tcomp,[I,V]);

```

**Ćwiczenie 12.4.** Czy teraz jesteś w stanie zbadać *zależność*  $y_N$  „przy  $t \rightarrow \infty$ ” dla różnych  $0 \leq \tau \leq 1$ ? Jaką możesz postawić hipotezę? Czy potrafisz ją udowodnić?

# Literatura

- [1] Nikolai Chernov. Fitting ellipses, circles, and lines by least squares, 2010. <http://www.math.uab.edu/~chernov/cl>.
- [2] Paul Kienzle David Bateman. Octave-forge home page. <http://octave.sourceforge.net>.
- [3] Peter Deuffhard, Andreas Hohmann. *Numerical analysis in modern scientific computing*, wolumen 43 serii *Texts in Applied Mathematics*. Springer-Verlag, New York, wydanie II, 2003. An introduction.
- [4] John W. Eaton. GNU Octave home page. <http://www.gnu.org/software/octave/>.
- [5] Urszula Forýs. A short course on dynamical systems in biomathematics. <http://www.mimuw.edu.pl/~urszula/course.pdf>.
- [6] M.G.F Fuortes, A.L. Hodgkin. Changes in time scale and sensitivity in the ommatidia of *Limulus*. *J. Physiol.*, 172:239–263, 1964. <http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1368830/pdf/jphysiol01198-0077.pdf>.
- [7] Walter Gander, Jiri Hrebicek. *Solving Problems in Scientific Computing Using Maple and MATLAB*. Springer, New York, wydanie IV, 2004.
- [8] Curtis F. Gerald, Patrick O. Wheatley. *Applied Numerical Analysis*. Addison Wesley, wydanie VII, 2003.
- [9] Maciej Gonet. *Excel w obliczeniach naukowych i technicznych*. Helion, Gliwice, 2009.
- [10] E. Hairer, S. P. Nørsett, G. Wanner. *Solving ordinary differential equations. I*, wolumen 8 serii *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, wydanie II, 1993. Nonstiff problems.
- [11] Michael T. Heath. *Scientific Computing: An Introductory Survey*. McGraw-Hill, New York, wydanie II, 2002.
- [12] Nicholas J. Higham. *Accuracy and stability of numerical algorithms*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, wydanie II, 2002.
- [13] Alan Hindmarsh, Radu Serban. User Documentation for CVODE 2.6.0, 2009. <https://computation.llnl.gov/casc/sundials/documentation/documentation.html>.
- [14] Alan Hindmarsh, Radu Serban, Carol Woodward. SUNDIALS (SUite of Nonlinear and Differential/ALgebraic equation Solvers). <https://computation.llnl.gov/casc/sundials/main.html>.
- [15] J.M. Jankowsky, M. Dryja. *Przegląd metod i algorytmów numerycznych, tom I i II*. Biblioteka inżynierii oprogramowania. Wydawnictwo Naukowo-Techniczne, Warszawa, 1995.
- [16] Z. Jericević, Z. Kuster. Non-linear optimization of parameters in Michaelis–Menten kinetics. *CCA-CAA*, 78(4):519–523, 2005.
- [17] Brian W. Kernighan, Dennis M. Ritchie. *Język ANSI C*. Kłasyka informatyki. Wydawnictwa Naukowo-Techniczne, Warszawa, 2000.
- [18] A. Kielbasiński, H. Schwetlick. *Numeryczna algebra liniowa*. Wydawnictwa Naukowo-Techniczne, 1992.
- [19] Piotr Krzyżanowski, Philippe Laurençot, Dariusz Wrzosek. Mathematical models of receptor-mediated transport of morphogens. *M3AS*, 2010.
- [20] Piotr Krzyżanowski. *Obliczenia inżynierskie i naukowe. Skuteczne, szybkie, efektywne*. PWN, 2011. <http://www.mimuw.edu.pl/~przykry/obliczenia>.
- [21] J.R. Marti, A.C. Soudack. Ferroresonance in power systems: Fundamental solutions. *IEE Proceedings-C*, 138(4):321–329, 1991. [http://137.82.61.1/~jrms/91-07\\_Ferroresonance.pdf](http://137.82.61.1/~jrms/91-07_Ferroresonance.pdf).
- [22] Ignacy Misztal. Computational techniques in animal breeding, 1999–2007. <http://nce.ads.uga.edu/~ignacy/course2002/notes.pdf>.
- [23] ParaView home page. <http://www.paraview.org>.
- [24] Michael A. Savageau. Development of fractal kinetic theory for enzyme-catalysed reactions and implications for the design of biochemical pathways. *BioSystems*, 47:9–36, 1998.

- 
- [25] L. F. Shampine, S. Thompson. Solving DDEs in MATLAB. *Appl. Numer. Math.*, 37(4):441–458, 2001.
  - [26] Kermit N. Sigmon. MATLAB primer, 1993. <http://web.mit.edu/6.777/www/downloads/primer.pdf>.
  - [27] Amy Squillacote. *ParaView Guide*. Kitware, Inc., 2008.
  - [28] William J. Stewart. A comparison of numerical techniques in Markov modeling. *Comm. ACM*, 21(2):144–152, 1978. <http://cacm.acm.org/magazines/1978/2/11253-a-comparison-of-numerical-techniques-in-markov-modeling/pdf>.
  - [29] William J. Stewart. *Introduction to the numerical solution of Markov chains*. Princeton University Press, Princeton, NJ, 1994.
  - [30] Tomasz Strabel. Genetyka cech ilościowych zwierząt w praktyce, 07.12.2006. <http://jay.up.poznan.pl/~strabel/dydaktyka/gci.pdf>.
  - [31] VTK — The Visualization Toolkit home page. <http://www.vtk.org/>.
  - [32] J. T.-F. Wong. *Kinetics of Enzyme Mechanisms*. Academic Press, New York, 1975.